Types of Objects - Float

- Real numbers based on the native architecture of the double precision point or floating representation

float <=> real → -1, 0, +1, or nil

Returns -1, 0, or +1 depending on whether `float` is less than, equal to, or greater than `real`. This is the basis for the tests in the `Comparable` module.

The result of `NaN <=> NaN` is undefined, so an implementation-dependent value is returned.

`nil` is returned if the two values are incomparable.

A float wont work the same on a mac and a linux system.

< = >
-1 -> first value is less than
0 -> equals
1 -> last one is lesser than the one from the left
nil -> when values are incomparable.

Types of Objects - Rational

- Values that can be expressed as a fraction

```
Rational(1)      #=> (1/1)
Rational(2, 3)   #=> (2/3)
Rational(4, -6)  #=> (-2/3)
3.to_r           #=> (3/1)
2/3r             #=> (2/3)
```

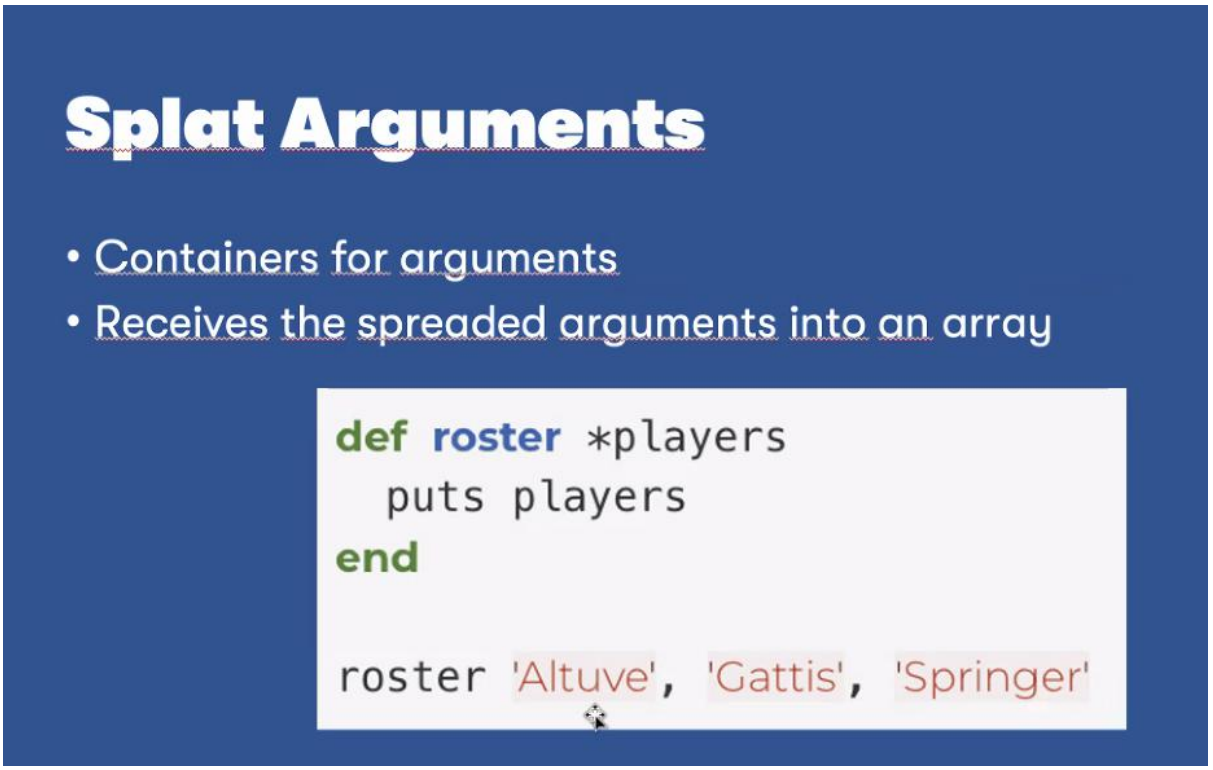
Arguments

Default arguments

```
def name( arg1, arg2, arg3, ... )
    .. ruby code ..
    return value
end
```

No optional nor splat arguments are gonna be processed first.
The required arguments are gonna be processed first.

Splat arguments



Usually placed at the end of the function at least we use keyboard splat arguments
Here we passed them as a list.
They are received as an array

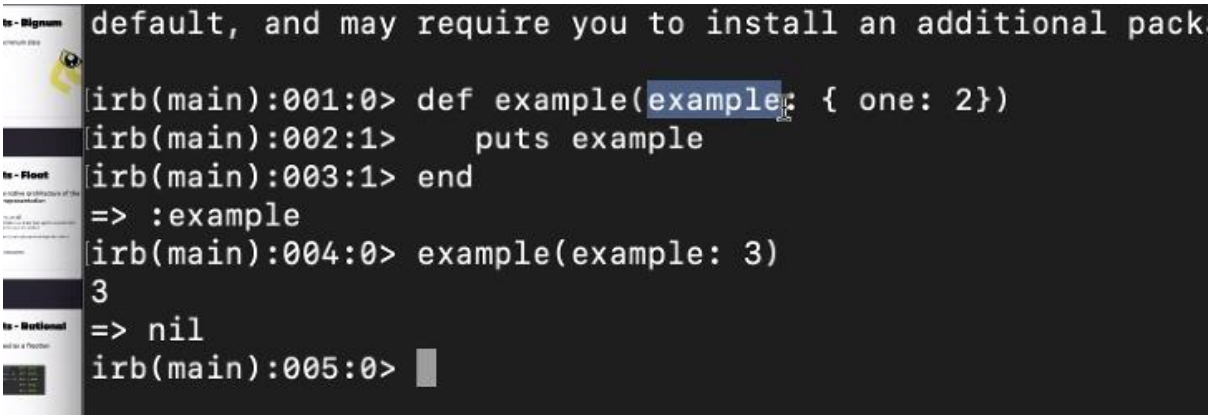
Keyword splat arguments

- These receives with two asterisks
- Always passed
  - Unlike optional arguments
- Required or empty, so it's a good practice to populate it whenever you start the method

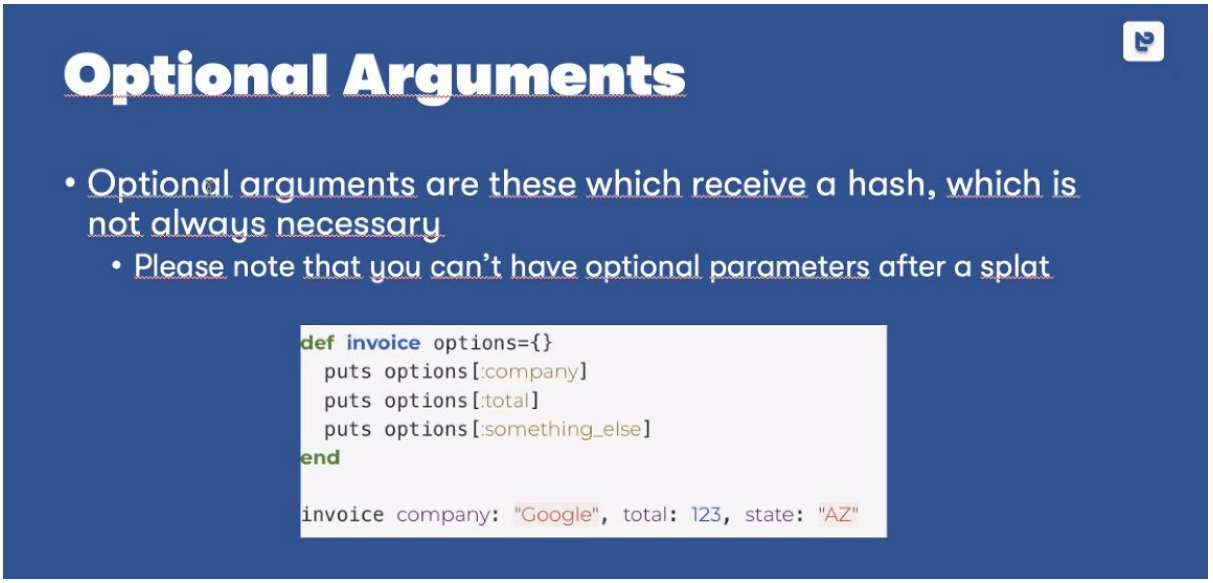
```ruby
def roster **players_with_positions
  players_with_positions.each do |player, position|
    puts "Player: #{player}"
    puts "Position: #{position}"
    puts "\n"
  end
end

data = {
  "Altuve": "2nd Base",
  "Alex Bregman": "3rd Base",
  "Evan Gattis": "Catcher",
  "George Springer": "OF"
}

roster data
```

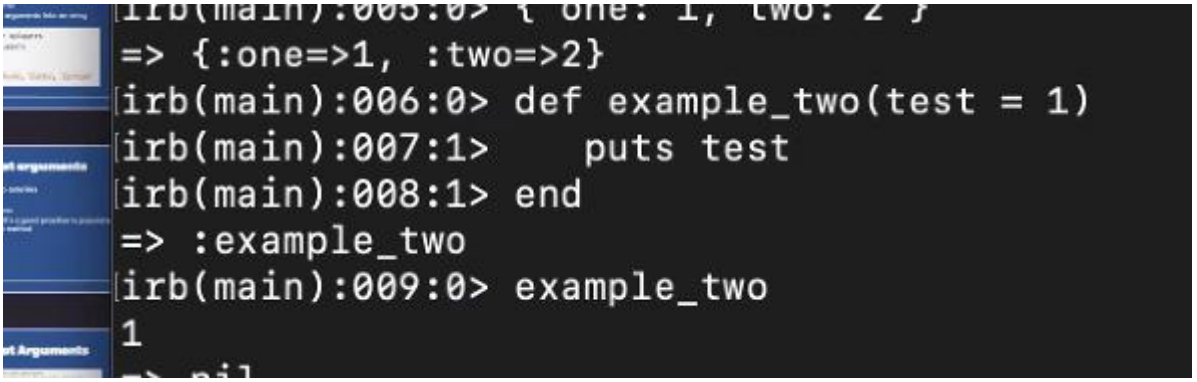
this arguments receives hashes

Keyword arguments



They are optional and almost always declared as a hash or as a symbol.





Last Line and Return

- The last line, even if it doesn't include the keyword return, is always returned in ruby

```
def multiply(val1, val2 )
     result = val1 * val2
     return result
end

value = multiply( 10, 20 )
puts value
```

Ranges

Inclusive and exclusive

Ranges

- A set of numbers or words between an starting point and an ending point.
- Inclsuive ..
- Exclusive ...

```
1..10    # Creates a range from 1 to 10
inclusive
1...10   # Creates a range from 1 to 9
```

Converting Range into Array

- Click to add text

```
(1..10).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

(1...10).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Using ranges in strings

- Get the rejected values of the Enumerable with .reject
- Know if it's included with .include?

```
words = 'cab'..'car'

words.min          # get lowest value in range
=> "cab"

words.max          # get highest value in
range
=> "car"

words.include?('can') # check to see if a
value exists in the range
=> true

words.reject {|subrange| subrange < 'cal'} #
reject values below a specified range value
=> ["cal", "cam", "can", "cao", "cap", "caq",
"car"]

words.each {|word| puts "Hello " + word} #
iterate through each value and perform a task
```

(1 .. 2) === 4 **si incluye o no el 5 en el range**

Ranges in logical expressions

- Click to add text

```
(1..20) === 15       # Does the number fit in the range 1 to
20
=> true
('k'..'z') === 'm'   # Does the letter fall between the
letters 'k' and 'z' in the alphabet?
=> true
```

Arrays

Most common way of declaring and instancing arrays

```
[irb(main):001:0> numbers = [1, 2, 3, 4, 5]
 => [1, 2, 3, 4, 5]
 irb(main):002:0>
```

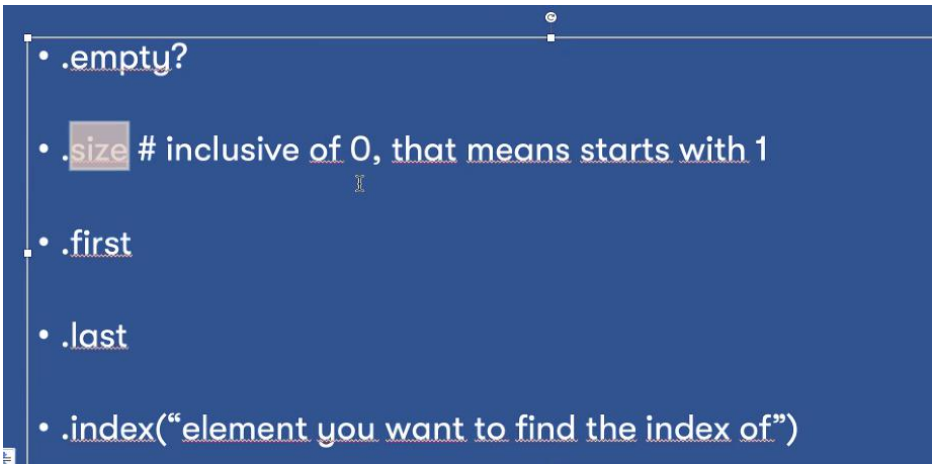
Arrays can start with 0 elements, and then add more.

We can add a default value

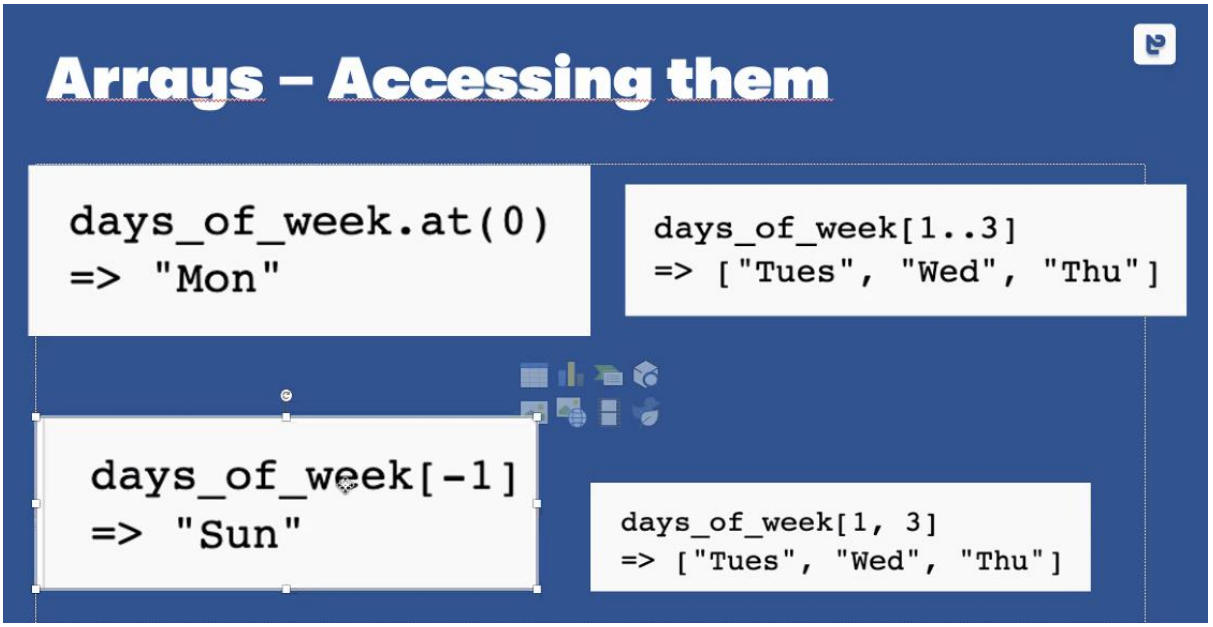
Click to add text

```
days_of_week = Array.new(7, "today")
=> ["today", "today", "today", "today", "today", "today",
"today"]
```

Common methods



Accessing arrays



.at and [] are the same

Arrays – Adding elements

- array.push("new value")
- Or:

```
days1 = ["Mon", "Tue", "Wed"]
days1 << "Thu" << "Fri" << "Sat" << "Sun"
=> ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"]
```

Arrays– Logical operators

Operator	Description
-	Difference - Returns a new array that is a copy of the first array with any items that also appear in second array removed.
&	Intersection - Creates a new array from two existing arrays containing only elements that are common to both arrays. Duplicates are removed.
|	Union - Concatenates two arrays. Duplicates are removed.

They are like diagrams of ben, they are elike joins in postgress.

pop -> removes and return last elements.

Modify

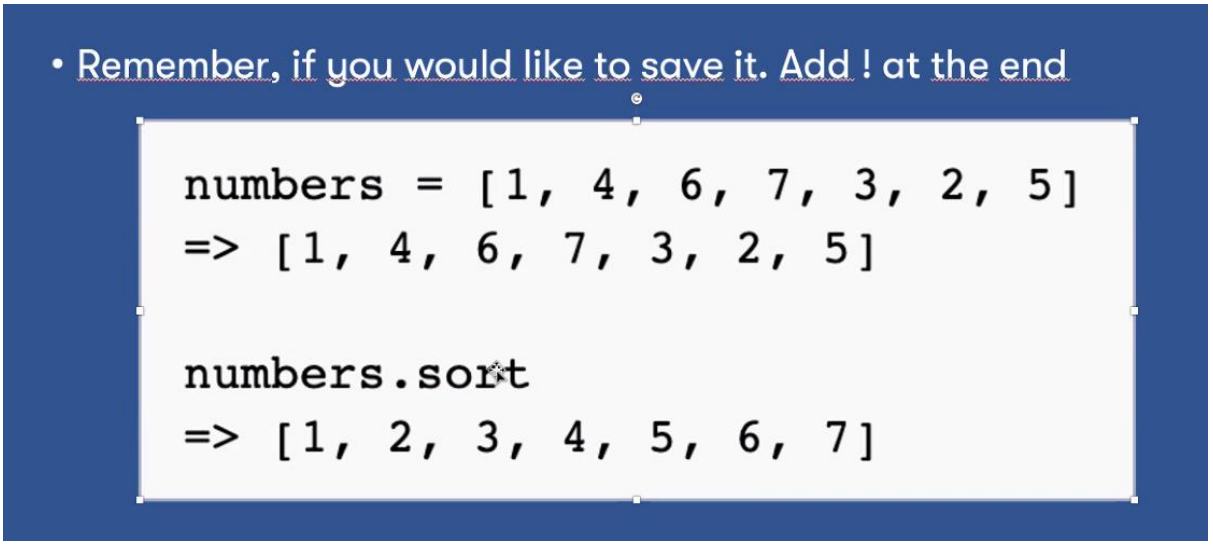
Most common way

```
colors[1] = "yellow"
=> "yellow"
```

Delete

```
colors.delete_at(1)
=> "green"
```

Sort



Operators

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators

Comparison

- Click to add

Comparison Operator	Description
==	Tests for equality. Returns *true* or *false*
.eql?	Same as ==.
!=	Tests for inequality. Returns *true* for inequality or *false* for equality
<	Less than. Returns *true* if first operand is less than second operand. Otherwise returns *false*
>	Greater than. Returns *true* if first operand is greater than second operand. Otherwise returns *false*.
>=	Greater than or equal to. Returns *true* if first operand is greater than or equal to second operand. Otherwise returns *false*.
<=	Less than or equal to. Returns *true* if first operand is less than or equal to second operand. Otherwise returns *false*.
<=>	Combined comparison operator. Returns 0 if first operand equals second, 1 if first operand is greater than the second and -1 if first operand is less than the second.

The triple equals behaves the same as the double equals

```
irb(main):012:0> a += 3
=> 6
irb(main):013:0> (3..2) === 1
=> false
irb(main):014:0> 3 === 3.0
=> true
```

But the triple one is used specially for ranges.

Bit Level Operations

- Click to add text

Combined Operator	Equivalent
~	Bitwise NOT (Complement)
|	Bitwise OR
&	Bitwise AND
^	Bitwise Exclusive OR
<<	Bitwise Shift Left
>>	Bitwise Shift Right

<< moves the elements to the left
>> the same but opposite way

Homework:
Reading the articles of today

Math methods are important also:

Math Methods

no siempre los keyboard arguments tienen valor, dehecho es buena practica darles valores pre-deteminados.