

METODOS DE ARREGLOS RUBY

AÑADIR

- ☐ unshift
- ☐ insert
- ☐ push

ELIMINAR

- ☐ pop
- ☐ shift
- ☐ delete
- ☐ delete_at
- ☐ delete_if
- ☐ keep_if

RECUPERAR ELEMENTOS

- ☐ first
- ☐ last
- ☐ at
- ☐ fetch
- ☐ take
- ☐ take_while

FILTROS

- ☐ select
- ☐ filter

AGREGAR ELEMENTOS



```
orden_jedi = [  
  "Qui-Gon Jinn",  
  "Mace Windu",  
  "Ki-Adi-Mundi",  
  "Obi-Wan Kenobi"  
]
```

```
orden_jedi.unshift('Joda')
```

```
orden_jedi.each do |jedi|  
  puts jedi.downcase  
end
```

El método `.unshift()` agrega un nuevo elemento al inicio.

```
orden_jedi = [  
  "Qui-Gon Jinn",  
  "Mace Windu",  
  "Ki-Adi-Mundi",  
  "Obi-Wan Kenobi"  
]
```

```
orden_jedi.insert(2, 'Joda')
```

```
orden_jedi.each do |jedi|  
  puts jedi.downcase  
end
```

El método `.insert()` agrega un nuevo elemento en una posición.

```
orden_jedi = [  
  "Qui-Gon Jinn",  
  "Mace Windu",  
  "Ki-Adi-Mundi",  
  "Obi-Wan Kenobi"  
]
```

```
orden_jedi.push('Joda')
```

```
orden_jedi.each do |jedi|  
  puts jedi.downcase  
end
```

El método `.push()` agrega un nuevo elemento al final.



```
heros_dc = [  
    "superman",  
    "green arrow",  
    "catwoman",  
    "shazam"  
]
```

```
heros_dc.shift
```

```
puts heros_dc
```

El método `.shift` elimina un elemento al inicio.

```
heros_dc = [  
    "superman",  
    "green arrow",  
    "catwoman",  
    "shazam"  
]
```

```
heros_dc.pop
```

```
puts heros_dc
```

El método `.pop` elimina un elemento al final.

```
heros_dc = [  
    "superman",  
    "green arrow",  
    "catwoman",  
    "shazam"  
]
```

```
heros_dc.delete_at(2)
```

```
puts heros_dc
```

El método `.delete_at` elimina un elemento en el índice.

```
heros_dc = [  
    "superman",  
    "green arrow",  
    "catwoman",  
    "shazam"  
]
```

```
heros_dc.delete('shazam')
```

```
puts heros_dc
```

El método `.delete` elimina todo elemento coincidente.

REMOVER ELEMENTOS CONDICIONALMENTE



```
numbers = [2,9,-4,-2,6,-7]
numbers.delete_if { |num| num < 0 }
puts numbers
```

El método `.delete_if` elimina todo elemento que cumpla la condición.

```
numbers = [2,9,-4,-2,6,-7]
numbers.keep_if { |num| num < 0 }
puts numbers
```

El método `.keep_if` es la contraparte de `.delete_if`, mantiene los elementos que cumplan la condición.



```
pares = [2,9,-4,-2,3,5,-7].select { |num| num.even? }  
puts pares
```

El método `.select` retorna un nuevo arreglo con todos los elementos que cumplan la condición.

```
letras = %w[ a b c d e f g h i j k l m n o p q r s t ]  
vocales = letras.filter { |letra| letra =~ /[aeiou]/ }  
puts vocales
```

El método `.filter` es un alias para `.select`, usar expresiones regulares es una excelente opción para una búsqueda (`=~` operador match).



```
heros_dc = [  
    "superman", "aquaman", "batman", "flash"  
]  
  
heros_marvel = [  
    "deadpool", "hulk", "wolverine", "ironman"  
]  
  
multiverse = heros_dc.concat(heros_marvel)  
puts multiverse
```



El método `.concat` retorna la unión de dos o más arreglos
(no modifica los arreglos involucrados).

TRANSFORMAR ARREGLOS



```
armaduras_de_bronces = [ "andromeda", "pegaso", "cisne", "dragon", "fenix" ]  
longitud_nombres = armaduras_de_bronces.map { |armadura| armadura.length }  
puts longitud_nombres
```



El método `.map` se utiliza para transformar un arreglo, retornando el arreglo modificado, llama a una función una vez por cada elemento del arreglo y así crea un nuevo arreglo (no modifica el arreglo original).

LIMPIAR ARREGLOS



```
pokemones = [
  "articuno", "charizard", "pidgey", "Articuno", "Pidgey"
]

pokemones_unicos = pokemones.uniq { |p| p.downcase }

puts pokemones_unicos
```



El método `.uniq` retorna un nuevo arreglo, removiendo los valores duplicados (el método con bang `.uniq!` retorna nil si no se hay cambios).

```
pokemones = [
  "muk", "charizard", "sunflora", "chikorita", "treecko"
]

tipo_planta = ["sunflora", "treecko", "chikorita"]

sin_tipo_planta = pokemones.difference(tipo_planta)

puts sin_tipo_planta
```



El método `.difference` retorna un nuevo arreglo, removiendo los valores que estén en otro arreglo.





```
numeros = [25,93,17,29,48,88,39,81,62,82,9,4,2,6,70]
revancha = Array.new(10) { rand(100) }.sample(5)
ganador = numeros.sample(5)
print "Ganador %s\nRevancha %s" % [ganador, revancha]
```



El método **.sample** retorna un nuevo arreglo con elementos elegidos aleatoriamente, si no le damos argumento, devuelve un solo elemento

BARAJAR LA POSICIÓN DE LOS ELEMENTOS EN ARREGLOS



```
poker = [ 1,2,3,4,5,6,7,8,9,10,"j","q","k","a" ]  
barajado = poker.shuffle  
print barajado
```

El método `.shuffle` retorna un arreglo con los elementos mezclados aleatoriamente

RECUPERAR UN ELEMENTO



```
digimones = [  
    "etemon",  
    "agumon",  
    "skullgreymon",  
    "leomon",  
    "angemon"  
]
```

```
print digimones.first
```

`.first` trae el primer elemento del arreglo.



```
digimones = [  
    "etemon",  
    "agumon",  
    "skullgreymon",  
    "leomon",  
    "angemon"  
]
```

```
print digimones.at(3)
```

`.at` trae el elemento en la posición dada.



```
digimones = [  
    "etemon",  
    "agumon",  
    "skullgreymon",  
    "leomon",  
    "angemon"  
]
```

```
print digimones.last
```

`.last` trae el último elemento del arreglo.

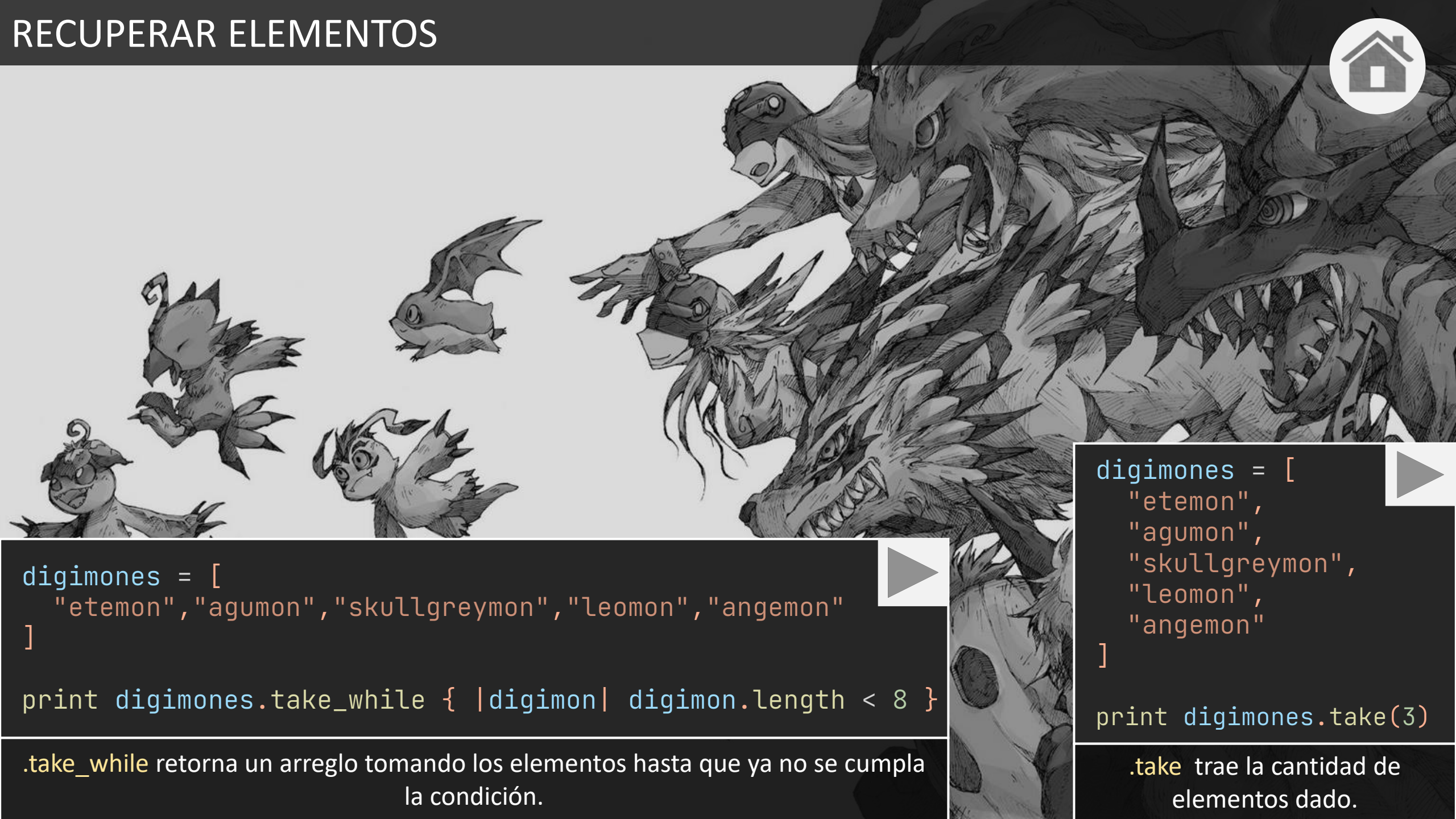


```
digimones = [  
    "etemon",  
    "agumon",  
    "skullgreymon",  
    "leomon",  
    "angemon"  
]
```

```
print digimones.fetch(3)
```

`.fetch` trae el elemento en la posición dada, error si no existe.





```
digimones = [  
    "etemon", "agumon", "skullgreymon", "leomon", "angemon"  
]
```

```
print digimones.take_while { |digimon| digimon.length < 8 }
```

`.take_while` retorna un arreglo tomando los elementos hasta que ya no se cumpla la condición.

```
digimones = [  
    "etemon",  
    "agumon",  
    "skullgreymon",  
    "leomon",  
    "angemon"  
]
```

```
print digimones.take(3)
```

`.take` trae la cantidad de elementos dado.



```
invitados = [  
    "juan", "alberto", "marco", "sandra", "eva"  
]
```

```
print invitados.include?("alberto")
```



`.include?` retorna un booleano, consulta si el argumento está dentro del arreglo.

