Glosario

- **Mutabilidad**: Se refiere a si un objeto (no confundir con una variable) puede cambiar. Es un concepto importante de manejar porque se pueden introducir errores inesperados.

- **Método mutable**: Un método que cambia el estado de un objeto.

```
a = [4, 2, 3, 1]
    a.sort! # El valor de a cambia
```

- **Método inmutable**: Un método que NO cambia el estado de un objeto.

```
a = [4, 2, 3, 1]
    a.sort # El valor de a No cambia
```

- **Método de instancia**: Un método que se llama desde una instancia específica.

```
class Foo
    def bar
        puts "Soy un método de clase"
    end
end

Foo.new.bar # funciona porque Foo.new es una instancia
Foo.bar # No funciona, porque Foo es una clase.
```

- **Operadores como métodos**: Los operadores como el `+` o el `-` son métodos de instancia de un objeto, nosotros podemos agregar nuevos métodos o redefinir los existente.

```ruby
class SuperIgual
    def ==(otrocosa)
        true
   end
end

puts SuperIgual.new == 2
puts SuperIgual.new == "hola"
puts SuperIgual.new == nil
```

Aquí; definimos una clase y le creamos un método para poder compararla con otras.

- **Método de clase**:Un método que se llama desde la clase de un objeto, se definen como `self.metodo` o `Clase.metodo`

```ruby
class Clase
    def self.metodo
        puts "Soy un método de clase"
    end
end
```

- **Variables de instancia**: Son las variables que definen el estado de un objeto, se definen con `@`

```ruby
class Persona
    def initialize
        @edad = 15
    end
end
```

- **Variables de clases**: Son las variables que definen el estado de la clase, se definen con `@@`

```ruby
class Persona
    @@edad_inicial = 0
    def initialize
        @edad = @@edad_inicial
    end
end
```

- **Encapsulamiento**: Esto quiere decir que las variables no se pueden acceder fuera de un objeto. Para hacerlo tenemos que ocupar métodos getter y setter.

```ruby
class Persona
    def initialize
        @edad = 15
    end
end

Persona.new.edad
#Error, porque @edad no es accesible desde fuera
```

- **self**: El keyword self nos da acceso al objeto actual.

Dentro de un mé;todo lo podemos ocupar (implícita o explícitamente) para llamar a otros métodos.

Dentro de la clase lo podemos ocupar para definir métodos de clase.

- **Self implícito vs explícito**: Cuando llamamos a un método dentro de una clase no es necesario anteponer `self.` ruby lo hace automáticamente por nosotros, pero si queremos podemos hacerlo a esto se le llama self explícito.

```ruby
class Foo
    def bar
    end

    def baz
        self.bar #llama a bar
        bar #llama a bar
    end
end
```

- **herencia**: La capacidad que tiene un objeto de heredar todos los atributos y métodos de otro, esto se hace para no repetir código y dentro de Rails se ocupa mucho, lo necesitaremos para saber donde agregar nuestro código.

- **super**: Es la forma de llamar al método padre de un objeto.

- **módulos**: Es otro forma de ordenar nuestro código, los mó;dulos hacen fácil la definición de constantes y además permiten agrupar diversas clases.

- **mixin**: Es la forma de incorporar los métodos que tiene un módulo dentro de una clase.