

DESAFIO **LATAM**

RUBY OBJETOS



Introducción a clases y objetos

En una clase anterior creamos un método para verificar elementos si un elemento estaba en un array

```
def check_if_exist(array, object)
  array.each do |element|
    if element == object
      return true
    end
  end
  false
end
```

Para ocuparlo:

```
check_if_exist(array,
objeto)
```

Pero el método include? lo llamamos así:

```
array.include?(objeto)
```

LOS OBJETOS SON GENIALES

MÁS MOTIVACIÓN

- Los objetos nos permiten:
 - Reutilizar y compartir código de forma fácil
 - Abstraernos de los datos para pensar en alto nivel
 - Necesitamos saber de objetos para poder crear aplicaciones con Rails

NUEVOS CONCEPTOS

Algunos conceptos que vamos a aprender hoy:

- Clase
- Objeto
- Instancia
- Atributo
- Método de clase y de instancia
- Variables locales, de clase y de instancia
- Getters y Setters
- Constructor
- Self

¿QUÉ ES UN OBJETO?

En ruby salvo pocas excepciones **todo** es un objeto, los literales como los Fixnum, Strings, son objetos, los Array, Hashes, y tanto True como False son objetos.



Idea de concepto simple:

Agrupemos el código en torno a entidades

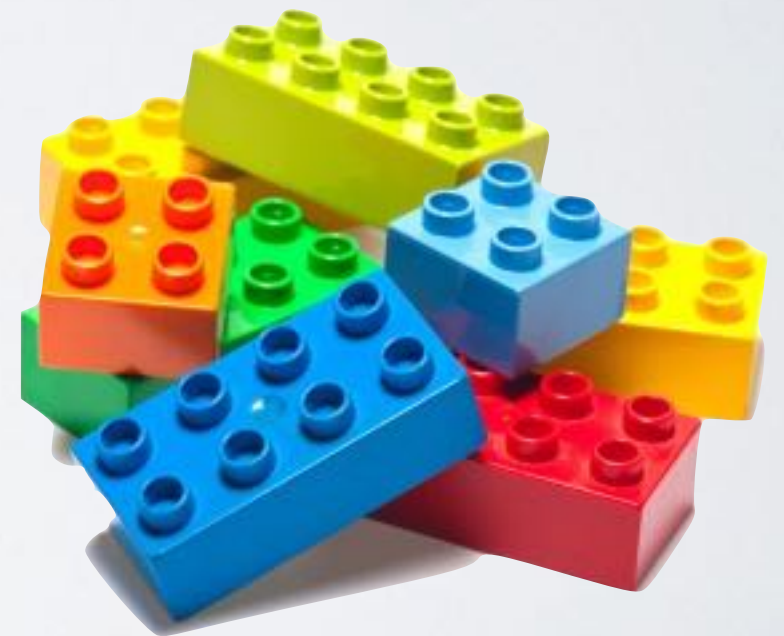
Para eso definimos una clase

LOS TRES CONCEPTOS MÁS IMPORTANTES



CLASE

“El secreto para comprender bien la programación orientada a objetos es entender a las clases como una fábrica o **molde de objetos** “



CLASE V/S OBJETO

La clase es el molde, el objeto es el producto, y un objeto específico creado de esa clase se le llama **Instancia**



“Por ejemplo un molde de una pieza de lego (la clase) y los objetos como los productos de este molde, o sea los legos “

CLASE EN RUBY

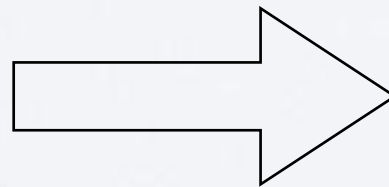
En ruby para definir una clase:

```
class Lego  
end
```

Para crear un objeto a partir de ese molde hay que **instanciarlo**, para **instanciar** basta con utilizar el nombre de la clase con el método **new**:

Objeto en la terminal:

```
lego1 = Lego.new
```



```
#<Lego:0x007ff7e0836b90>
```

RESUMEN

- Las **clases** son un molde que uno programa.
- Los **objetos** son los productos del molde.
- A un objeto específico de una clase se le llama **instancia**
- A través del **método new** creas instancias

EJERCICIO

Crea la clase lego e instancia 10 objetos legos guardándolos en un array

CASI TODOS LOS OBJETOS SE
INSTANCIAN CON .NEW

PERO ALGUNOS QUE YA CONOCEMOS
TIENEN OTRA FORMA DE
INSTANCIARSE

```
a = Array.new()
```

=>

```
a = []
```

```
b = Hash.new()
```

=>

```
b = {}
```

PARA SABER DE QUE CLASE ES UN
OBJETO PODEMOS UTILIZAR EL
MÉTODO .CLASS

```
a = []  
a.class
```

```
b = {}  
b.class
```


OBJETOS TIENEN:

- 1) Identidad
- 2) Comportamiento
- 3) Atributos

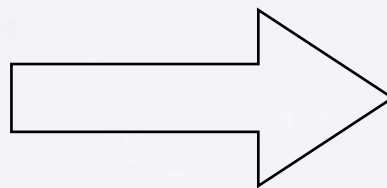
“Entonces tenemos la clase que es un molde de legos, que es capaz de imprimir objetos lego, los cuales tienen un **identificador** (por ejemplo un número de serie), **atributos** como el color y tamaño; y **comportamientos** como acoplarse y causar dolor infinito si uno los pisa.”

IDENTIDAD

En ruby todos los objetos tienen un identificador, el lego que creamos previamente tiene uno, este se puede capturar utilizando el método `object_id`

Id en la terminal: (ejemplo)

```
lego1.object_id
```



```
#=>70351300158900
```

COMPORTAMIENTO

El comportamiento define que puede hacer un objeto

Podemos definir comportamientos agregando método dentro de una clase.

```
class Zombie
  def saludar
    puts "braaaains"
  end
end
```

Este tipo de método recibe el nombre de **métodos de instancia**

OCUPANDO UN MÉTODO DE INSTANCIA

```
class Zombie
  def saludar()
    puts "Braaaains"
  end
end

z = Zombie.new
z.saludar() # => Braaaains
```

Creamos una instancia
Llamamos al método de instancia

ATRIBUTOS

```
class Ejemplo
  def guardar_edad(edad)
    @edad = edad
  end
end
```

Método de instancia

Atributo

Los atributos son las variables de instancias, estas se identifican porque empiezan con @

LAS VARIABLES DE INSTANCIA DEPENDEN DE LA INSTANCIA

```
class Persona
  def bautizar(nuevo_nombre)
    @nombre = nuevo_nombre
  end

  def saludar()
    puts "#{@nombre} dice hola"
  end
end

p1 = Persona.new
p1.bautizar("Jaime")
p2 = Persona.new
p2.bautizar("Francisca")
p2.saludar()
```

¿Qué mensaje se mostrará en pantalla?

VARIABLES DE INSTANCIA SON DISTINTAS A LAS VARIABLES LOCALES

```
class Foo
  def bar
    dato = 10
    @otro_dato = 20
  end
end
```


PARA ENTENDER LA DIFERENCIAS NECESITAMOS DEFINIR UN NUEVO CONCEPTO

Ámbito (Scope)

Es el contexto que tiene un nombre dentro de un programa.
El ámbito determina en qué partes del programa una entidad
puede ser usada.

HASTA EL MOMENTO HEMOS TRABAJADO EXCLUSIVAMENTE CON VARIABLES LOCALES

El ámbito de las variables locales es siempre una de las siguientes

```
proc{ ... }  
loop{ ... }  
def ... end  
class ... end  
module ... end  
the entire program (unless one of the above applies)
```

```
proc{ ... }  
loop{ ... }  
def ... end  
class ... end  
module ... end  
the entire program (unless one of the above applies)
```

- sea que si definimos una variable local dentro de un método, después del método esta no existirá

VARIABLES LOCALES VS DE INSTANCIA

```
1 class Ejemplo
2   def valores_iniciales()
3     @yo_soy_una_var_de_instancia = 5
4     yo_soy_una_var_local = 5
5   end
6   def mostrar_valores()
7     puts @yo_soy_una_var_de_instancia
8     puts yo_soy_una_var_local
9   end
10 end
11
12 ejemplo = Ejemplo.new()
13 ejemplo.valores_iniciales
14 ejemplo.mostrar_valores
```

Esto causará error porque la variable no **yo_soy_una_var_local** fue definida en otro método, en `mostrar_valores` no existe

VARIABLES LOCALES VS DE INSTANCIA

```
1 class Ejemplo
2   def valores_iniciales()
3     @yo_soy_una_var_de_instancia = 5
4     yo_soy_una_var_local = 5
5   end
6   def mostrar_valores()
7     puts @yo_soy_una_var_de_instancia
8     puts yo_soy_una_var_local
9   end
10 end
11
12 ejemplo = Ejemplo.new()
13 ejemplo.valores_iniciales
14 ejemplo.mostrar_valores
```

Esto causará error porque la variable no **yo_soy_una_var_local** fue definida en otro método, en `mostrar_valores` no existe

```
NameError: undefined local variable or method `yo_soy_una_var_local' for
#<Ejemplo:0x007fb4591080C8 @yo_soy_una_var_de_instancia=5>
Did you mean? @yo_soy_una_var_de_instancia
from (irb):30:in `mostrar_valores'
from (irb):36
from /Users/gonzalosanchez/.rvm/rubies/ruby-2.3.0/bin/irb:11:in `<main>'
2.3.0 :037 >
```

VEAMOS UN EJEMPLO

```
class PersonaX
  def empezar
    @edad = 1
  end

  def envejecer
    @edad += 1
  end

  def mostrar_edad()
    puts @edad
  end
end

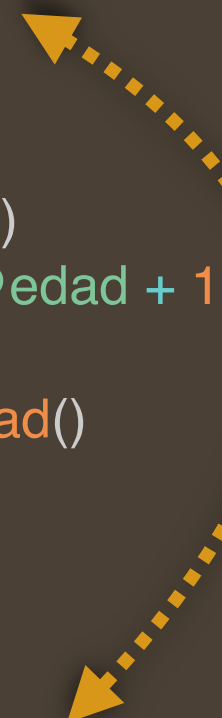
p1 = Persona.new
p1.empezar
p1.envejecer
p1.envejecer
p1.envejecer
p1.mostrar_edad
```

El método constructor

Initialize

```
class Persona
  def initialize()
    @edad = 1
  end
  def envejecer()
    @edad = @edad + 1
  end
  def mostrar_edad()
    puts @edad
  end
end


p1 = Persona.new
p1.envejecer
p1.envejecer
p1.envejecer
p1.mostrar_edad
```

A dashed yellow arrow originates from the 'p1 = Persona.new' line in the code block and points to the 'def initialize()' method, illustrating that the new method calls the initialize method to set up the object's state.

UN EJEMPLO CON ARGUMENTOS

```
class Persona
  def initialize(edad)
    @edad = edad
  end
  def envejecer()
    @edad = @edad + 1
  end
  def mostrar_edad()
    puts @edad
  end
end

p1 = Persona.new 5
p1.envejecer
p1.mostrar_edad
```

A dashed yellow arrow curves from the argument '5' in the 'new' method call to the 'edad' parameter in the 'initialize' method definition, illustrating how the argument is passed to the method.

EN RUBY LAS VARIABLES DE INSTANCIA NO SON ACCESIBLES DESDE FUERA DEL OBJETO (POR EJEMPLO EN MAIN)

```
class Foo
  attr_accessor :a
  def initialize
    @a = 5
  end
end

Foo.new.a
# Este es un error
# No se puede llamar a una
variable de un objeto en ruby
```

Sin getters y setters
no podemos acceder a @a

A este comportamiento se le llama encapsulamiento

¿CUÁL ES EL ERROR?

```
class Perro
  def initialize(edad)
    @edad = edad
  end
end

perro1 = Perro.new 10
puts perro1.edad
perro1.edad = 8
```

LOS GETTERS Y SETTERS SE PUEDEN CREAR DE FORMA AUTOMÁTICA CON ATTR_ACCESSOR

Pero primero aprenderemos
otra forma que nos enseñará mucho de ruby

GETTERS / SETTERS

Para acceder a los estados tenemos que crear métodos para obtener los estados (getters) y métodos para cambiarlos (setters)

getter

```
class Fraccion
  def initialize(a, b)
    @numerador = a
    @denominador = b
  end
```

```
  def get_numerador()
    @numerador
  end
```

El retorno es
implícito

```
  def set_numerador(valor)
    @numerador = valor
  end
end
```

setter

```
f1 = Fraccion.new(2,3)
puts f1.get_numerador()
f1.set_numerador(5)
puts f1.get_numerador()
```

GETTER Y SETTERS

VERSIÓN 2.0

```
class Perro
  def initialize(edad)
    @edad = edad
  end

  def edad
    @edad
  end

  def edad=(edad)
    @edad = edad
  end
end

perro1 = Perro.new 10
puts perro1.edad # Esto ahora si funcionará
perro1.edad = 8 # Esto también funcionará
```

ATTR_ACCESSOR

(hacer accesible el atributo)

Es posible definir de forma simultánea la variable, los getters y setters a través del **attr_accessor**:

```
class Perro
  attr_accessor :edad
  def initialize(edad)
    @edad = edad
  end
end

perro1 = Perro.new 10
puts perro1.edad
perro1.edad = 8
```

QUIZ RÁPIDO

Dado el siguiente código

```
class Alumno
  def initialize()
    @notas = []
    nombre = "Humberto"
  end
end
```

¿Cuál es el error con?

```
Alumno.notas
Alumno.new.notas
Alumno.nombre
Alumno.new.nombre
```

MODIFICAR EL CÓDIGO PARA PODER ACCEDER AL NOMBRE Y MODIFICAR LAS NOTAS

```
class Alumno
  def initialize()
    @notas = []
    nombre = "Humberto"
  end
end
```


¿CUÁL ES EL PROBLEMA CON EL SIGUIENTE CÓDIGO?

```
class Store
  attr_accessor :name
  def initialize(name)
    name = name
  end
end
```

```
store = Store.new("S1")
puts store.name
```

¿CUÁL ES EL ERROR CON EL SIGUIENTE CÓDIGO?

```
class Product
  attr_reader :stock
  def initialize
    @stock = 0
  end
end
```

```
p1 = Product.new
p1.stock = 5
```

AGREGAR UN MÉTODO DE INSTANCIA A **STUDENT** PARA CALCULAR EL PROMEDIO DEL ALUMNO

```
class Student
  def initialize(grade1, grade2, grade3)
    @grade1 = grade1
    @grade2 = grade2
    @grade3 = grade3
  end
end
```

```
student = Student.new(2,3,4)
student2 = Student.new(4,5,6)
```

EJERCICIO

- Crear la clase carta con número y pinta.
- Instanciar 5 cartas al azar.

¿CUÁL ES EL ERROR?

```
class Foo
  def method1
    puts "hola"
  end
end
```

```
Foo.method1
```

¿AQUÍ HAY 2 ERRORES GRANDES, CUÁLES SON?

```
class Foo(x,y)
  def initialize()
    puts "hola"
  end
end
```

```
Foo.new().method1(2,3)
```

REFLEXIONANDO SOBRE LOS MÉTODOS Y EL ATTR_ACCESSOR

No debemos confundir los métodos con las variables de instancias

```
class Foo
  attr_accessor :a
  def initialize
    @a = 5
    puts @a # Yo soy una variable
    puts a # Yo soy una método
  end
end

Foo.new.a # Yo estoy llamando al método

# No se puede llamar a una variable de un objeto en ruby
```

¿CUÁLES SON LOS VALORES?

```
class Foo
  attr_accessor :a
  def initialize
    @a = 5
  end

  def change
    a = 10
  end
end
```

```
foo = Foo.new
puts foo.a
puts foo.change
puts foo.a
```


OBJETOS Y ARREGLOS

```
class Product
  def initialize(name, stock = 10)
    @name = name
    @stock = 10
  end
end
```

```
products = []
10.times do |i|
  Product.new "product#{i}"
end
```

ESTRATEGIA PARA RESOLVER PROBLEMAS

Antes de resolver un problema donde haya código en main y en un objeto, preguntarse, ¿dónde debería ir el código?

Si queremos calcular el stock total de productos
donde debería ir el código

```
class Product
  def initialize(name, stock = 10)
    @name = name
    @stock = 10
  end
end
```

```
products = []
10.times do |i|
  Product.new "product#{i}"
end
```

Calcular el stock de varios productos no es problema de un
producto

Si es un script pequeño o una prueba podemos ponerlo en main

```
products = []  
10.times do |i|  
  Product.new "product#{i}"  
end
```

- podemos ponerlo en otro objeto, por ahora como ejercicio en clases lo pondremos en main

EJERCICIO

- Crear la clase alumno, cada alumno tiene un arreglo de calificaciones (enteros) y un nombre.
- Crear un arreglo con al menos 4 alumnos, cada alumnos.

Se pide: Calcular el promedio de notas del curso
Encontrar al alumno que tiene le promedio de notas más alto y devolver su nombre.