OOP

~~PURE~~ OBJECT ORIENTED PROGRAMMING

@mracos

- Object oriented programming is not **only** about **classes**!!
  - For instance, in javascript and lua we have prototype based inhiterance (no classes)
  - will explain later 😼

Object oriented programming is about: PASMEN,
objects

Remeber objects? That little things that you say that are **only** instantiated classes?

Well, they are much more

Take the red pill neo

Objects were made to simplify and abstract some concepts like

- state (data)
- ~~complex~~ interactions between and using *state*

One example:

- we have an `Account` object, with a `bankroll` state of 100 dollars
- we also want to have the possibility of `withdrawing` and `depositing` money

```ruby
account = Object.new
# => #<Object:0x00005645237b8810>
account.instance_eval do
    def bankroll
        @bankroll
    end
    def deposit(quantity)
        if @bankroll
            @bankroll += quantity
        else
            @bankroll = quantity
        end
    end
    def withdraw(quantity)
        @bankroll -= quantity
```

```
account.bankroll
# => nil
account.deposit 12
# => 12
account.bankroll
# => 12
account.withdraw 4
# => 4
account.bankroll
# => 8
```

- See that we didin't touch about `classes`, `inhiterances`, `methods`, `properties`, `polymorphism` and all that things?

*And yet, we covered the **OOP core**, that is **state and interactions** regarding objects*

- With **state** we can relate to `instance properties`
- As for **message passing**, here we can call then `methods`

- we are passing the `deposit` message to the account object
- with the 100 "body"

```
account.deposit 100
# => 108
```

- we have the bankroll state

```
account.instance_variables
# => [:@bankroll]
account.instance_variable_get "@bankroll"
# => 108
```

But why call then that? Because at least on more pure OOP languages, that is exactly what it means

When we call the objects methods we are *passing a message* to that object, we do **operations on properties to mess with the state**

We don't mess with the state directly!!

```ruby
class Account
    @bankroll
end
# => nil
account = Account.new
# => #<Account:0x000055b5abac0b00>
account.bankroll
# => NoMethodError: undefined method `bankroll' for #<Account:
account.bankroll = 12
# => NoMethodError: undefined method `bankroll=' for #<Account
```

```ruby
class Account
    def bankroll
        @bankroll
    end
    def bankroll=(bankroll)
        @bankroll = bankroll
    end
end
# => :bankroll=
```

- The accessor is a method

```
account = Account.new
# => #<Account:0x000055b5ab5a9f68>
account.bankroll
# => nil
```

- Even the assignment of a instance is a method!!

```
account.bankroll = 12
# => 12
account.bankroll
# => 12
```

- can be like this too

```
account.bankroll= 52
# => 52
account.bankroll
# => 52
```

To a language be *pure* OOP (like smalltalk, and on some points ruby) they have to have some properties like [1]

- **EVERYTHING** is an object
- no primitives like: `int`, `float` (they also are a class)
- objects **ONLY** communicate by message passing
  - that mean no operators
  - no public properties
- ...

- Everything is an object

```
number = 1
# => 1
string = "marcos"
# => "marcos"
number.class
# => Integer
string.class
# => String
```

```
1.class
# => Integer
"marcos".class
# => String
true.class
# => TrueClass
```

- A class is also an object!!! MINDBLOWN

```
class A
end
# => nil

A.is_a? Object
# => true
A.class
# => Class
```

- We only communicate by message passing

```ruby
1.positive?
# => true
1.methods.slice(18, 6)
# => [:==, :===, :[], :inspect, :size]
1.== 2
# => false
1.size
# => 8
1.respond_to? "+"
# => true
```

- not so pure

```
if.class
# => SyntaxError: unexpected '.'
```

Questions?

Thanks! 

EXTRAS

1. Prototype based inheritance