IRON
HACK

# Metaprogramming

Here Be Dragons

## Metaprogramming

Code that changes classes, instances, methods…

# Open classes

In Ruby all classes are open. This means we can add, remove or change methods in runtime

```ruby
class String
  def say_yay!
    puts "Yaaaaaaaaaaay!"
  end
end

"Oh, really?".say_yay!
```

## Open classes

```
puts Date.today
```

↓

NoMethodError: undefined method ‘today’ for Date:Class

# Open classes

```
require 'date'
puts Date.today
```

```
2014-09-18
```

# Exercise

Add a method called **salute** to all objects in Ruby that, once called, prints "Hello, I'm a <class name here>" in the screen.

Hint: Is there a common ancestor for all objects in Ruby?

## Monkey patching

If it walks like a duck and talks like a duck, it's a duck, right? So if this duck is not giving you the noise that you want, you've got to just punch that duck until it returns what you expect.

Monkey patching is often a bad idea

# Example

Cause ruckus in Pry by replacing the + operation in Fixnum

```
class Fixnum
  def +(number)
    rand * number
  end
end

puts 12 + 4
```

## Rails monkey patches classes extensively

## (small intermission)

## The splat*

```ruby
def my_method(*args)
  puts "The args are in an: #{args.class}"
  puts "You've sent: #{args.size} parameters"
  puts "The first one is: #{args[0]}"
end
```

(end of intermission)

# Method missing

```ruby
class Dummy
  def method_missing(m, *args, &block)
    puts "There's no method called #{m} here -- please try again."
  end
end

Dummy.new.whaaaat
```

# Exercise

Create an object that acts as a Hash but instead of using the "[", "]" operators you access the key name as a method.

To set value you just need to call the method with an argument

```
m = MagicObject.new
m.thing 2
puts m.thing # => 2
```

## Solution

```ruby
class MagicObject
  def initialize
    @data = {}
  end

  def method_missing(name, *args, &block)
    if args.size > 0
      @data[name] = args.first
    else
      @data[name]
    end
  end
end
```

## Calling methods

```
string = "Hi, I am a string"
string.send(:reverse)
# => "gnirts a ma I ,iH"
```

## Creating methods

```
class Object
  def self.say(something)
    define_method "say_#{something}".to_sym do
      puts something.upcase
    end
  end
end

class A
  say :yeah
end

A.new.say_yeah
```

→ YEAH

## Example

Can you create your own version of attr_accessor?

```ruby
def accessor(attribute_name)
  define_method attribute_name.to_sym do
    instane_variable_get "@#{attribute_name}".to_sym
  end

  define_method "#{attribute_name}=".to_sym do |value|
    instance_variable_set "@#{attribute_name}".to_sym, value
  end
end

class Thing
  accessor :attribute
end

t = Thing.new
t.attribute = "value"
t.attribute # => value
```

## Exercise

Create a method called shoutable so you can mark a method of a class as shoutable.

This creates a method in the class so you can call shout_<name of method> and you will get the output converted to string and upcased.

## Evaluate code

```
require 'date'
my_code = "Date.today"
eval my_code
```

## Exercise

Can you write your naïve implementation of Pry?