

**Alphorm**.com

Présentation de la formation

Le langage **Ruby**

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby**alphorm.com™©**

Plan

- Présentation du formateur
- Publics concernés
- Qu'est-ce que Ruby?
- Pourquoi Ruby?
- Le plan de formation
- Matériel requis
- Connaissances requises
- Liens des ressources utiles

**Le langage Ruby****alphorm.com™©**

Qui suis-je?

Yacine PETITPREZ

- 27 ans
- Développeur Ruby On Rails Fullstack (5 ans)
- Gérant Red Tonic Solutions, société d'outsourcing Ruby On Rails basée à Manille (Philippines)
- Mes références :
 - Mon profil **LinkedIn** : <http://fr.linkedin.com/in/anykeyh>
 - Mon profil **Alphorm** : <http://www.alphorm.com/auteur/yacine-petitprez>
- Me contacter:
 - yacine@redtonic.net
 - Les commentaires alphorm.com ☺



Publics concernés

- Développeurs Web
- Passionnés
- Porteurs de projet, fondateurs de *start-up*
- Les curieux

Qu'est-ce que Ruby?

- Langage de programmation créé en 1995 par Yukihiro Matsumoto (« Matz »)
 - Langage scripté (pas de compilation)
 - Fortement orienté objet: Tout est objet!
 - Fonctionnel
 - Syntaxiquement proche de l'anglais
- Largement popularisé par « Ruby On Rails », framework MVC de développement d'applications Web complètement écrit en Ruby.



Pourquoi Ruby?

- Langage facile à prendre en main. Idéal pour créer des scripts rapidement
- Formidable écosystème: **RubyGems**.
- Langage qui plait aux programmeurs: Clair, concis, multiple paradigmes de programmation possible.
- Si vous désirez créer un site ou une application web personnalisé de zéro.
 - Idéal pour les jeunes entrepreneurs du web qui ont des idées mais pas les ressources pour les faire produire par quelqu'un d'autre ☺
- En faire son métier: Salaires élevés et demande importante! (Et langage sympa)

Ils sont Ruby

- Scribd
- Twitter
- Groupon
- Basecamp
- Github
- DropBox
- En outils internes chez:
 - BBC, Nasa, New York Times etc etc...

Plan de la formation

- Chapitre 1: Préparer et être productif
 - Installer un environnement de développement Ruby
 - Créer notre premier script Ruby et présentation du projet

Plan de la formation

- Chapitre 2: Les bases du langage
 - Boucles et conditions
 - Les tableaux
 - Les hashes
 - Les fonctions
 - Le fonctionnel
 - Les modules
 - Les classes 1ère partie
 - Les classes 2nde partie

Plan de la formation

- Chapitre 3: Ruby avancé
 - Installer et utiliser des *gems*
 - Eclater le code dans plusieurs fichiers
 - Utilisation du Bundler
 - La réflexion du langage
 - Les DSL
 - Réouverture de classe et monkey patching
 - Finaliser le projet

Matériel requis

- Un ordinateur, si possible sous environnement Unix
 - Le must: Macbook Pro
 - Très bien aussi: Linux Ubuntu / Linux Mint
- Une machine virtuelle fait très bien l'affaire!!!
 - J'ai travaillé deux ans sur un notebook *Windows* avec *VirtualBox* et *Linux Mint* sans problèmes.
- Possibilité de développer sous *Windows*, mais je ne conseille pas (usage intensif de la console).

Connaissances requises

- Afin de profiter intégralement de ce cours, vous devez avoir quelques notions:
 - Notions dans l'usage du terminal sous Linux ou OSX
 - Connaître un minimum le développement informatique. Par exemple avoir développé en Javascript/HTML est un bon début.
- Ces points sont un plus non négligeable:
 - Savoir développer avec un langage orienté objet ou fonctionnel
 - Savoir utiliser linux de manière avancé
- Pas de panique pour les plus débutants: nous reprenons depuis le début!

Liens des ressources utiles

- La documentation de ruby-lang:
 - <https://www.ruby-lang.org/fr/documentation/>
- Ruby Warrior: Coder en ruby en s'amusant!
 - <https://www.bloc.io/ruby-warrior>
- De nombreuses personnes veulent vous aider, ne restez pas seul!
 - <http://www.meetup.com/parisrb/>
 - Il existe aussi des meetings dans toute les grandes villes de province!

Le langage Ruby

alphorm.com™©

Les autres formations dév sur Alphorm



Le langage Ruby

alphorm.com™©

Les autres formations dév web

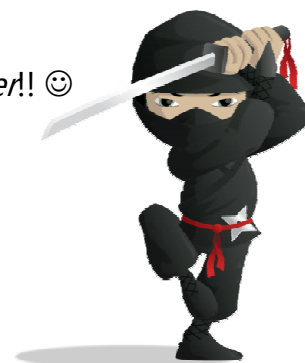


Le langage Ruby

alphorm.com™©

A la fin de la formation

- À la fin de la formation, vous aurez acquis:
 - Les concepts du langage Ruby – basiques et avancés
 - Créer une application dans un terminal ou un petit site avec Sinatra.
 - Prêt pour voir plus gros avec Ruby On Rails!
 - En bonne voie pour votre 1^{er} Dan de *Ruby Ninja Coder*!! 😊



Le langage Ruby

alphorm.com™©

 Are you ready ? 😊



Le langage Ruby

alphorm.com™©



Alphorm.com

Présentation de la formation

Installer un environnement
de développement Ruby

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

Plan de la vidéo

- Installation de RVM (*ruby version manager*)
 - Permet de gérer les différentes versions de ruby pour chaque projets.
- Installation d'un IDE: Sublime text
- Test de notre premier script ruby

Conclusion

- RVM: Contrôler la versions de Ruby
 - Optionnel mais bonne pratique
 - Permet d'avoir plusieurs version de Ruby sur le même ordinateur
 - Permet de s'assurer par exemple que chaque développeur de votre équipe travaille exactement sur la même version du langage.
 - *Gemset*. Gère différentes version des gems (bibliothèque de Gem)
 - No stress: Nous verrons plus tard les gems!



Conclusion

- Choisir votre éditeur de code:
 - Sublime text: Éditeur de code minimaliste mais puissant. Payant mais utilisable gratuitement.
 - Installer le package manager (aller dans la console et coller le code du package manager)
 - Vous pouvez naviguer dans les plugins afin de sélectionner ceux qui vous plaisent!
 - Autres alternatives: Vi, Gedit, RubyMine, Aptana, NetBeans...



Conclusion

- Lancer un code Ruby

```
ruby fichier.rb
```

- OU utiliser le shebang:

```
#!/usr/bin/env ruby
```

Pensez à rendre le fichier exécutable!

```
chmod +x monfichier.rb
```

- La commande « irb » permet de tester ruby de manière interactive.
- Afficher un message dans la console

```
puts "Bonjour Monde :)"
```

Aller plus loin

- <http://rvm.io>
 - Site officiel de RVM. Toutes les informations sur les possibilités du gestionnaire de version ruby.
- IDE principalement utilisés:
 - Sublime text: <http://www.sublimetext.com/>
 - RubyMine: <https://www.jetbrains.com/ruby/>
 - Netbeans: <https://netbeans.org/features/ruby/index.html>
 - Aptana: <http://www.aptana.com/products/radrails.html>

Ce qu'on a couvert

- Installation de RVM (*ruby version manager*)
- Installation d'un IDE: Sublime text
- Test de notre premier script ruby





Alphorm.com

Présentation de la formation

Présentation du projet et
création de notre premier
script ruby

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©



Plan de la vidéo

- Pourquoi faire un projet?
- Découverte du projet
- Pratique: Premier programme Ruby on Rails
- À retenir

Le langage Ruby

alphorm.com™©

Pourquoi faire un projet?

- Appliquer directement vos connaissances avec un but final
- Voir les étapes de réflexion du développeur dans la construction d'une application de manière itérative
- Donner un exemple de choses à faire qui peut vous être utile!
- Cet exemple n'est pas figé : Vous pourrez imaginer vos améliorations et changements personnels selon vos goûts!

ALORS, PRET À DECOUVRIR LE PROJET? 😊

Découverte du projet

- Création d'un projet de gestion de tâches à faire/fait
- Sera un outil qui vous permettra de lister vos tâches, les organiser/les supprimer:
 - `taskman list`
 - `taskman add "Apprendre ruby"`
 - `taskman mod 1 flag:important date:demain`
 - `taskman del 1`
- Les tâches seront sauvegardées dans un fichier structuré.



Ok, trêve de théorie, place à la pratique!



À retenir

- La liste des arguments passés à votre programme se situe dans la constante **ARGV**.
 - **ARGV** est un tableau!
 - Les **constantes** sont écrites en **MAJUSCULE** en Ruby. Une constante ne peut pas être réassignée à l'inverse d'une variable.
- Un outil puissant pour déboguer est la méthode **inspect** qui affiche l'objet
 - `p obj` #fait la meme chose que `puts obj.inspect`
- On accède aux arguments d'un tableau via l'opérateur **[]** et au nombre d'éléments via la méthode **count**

Ce qu'on a couvert

- Découverte du projet
- Premier programme Ruby on Rails



Le langage Ruby

alphorm.com™©



Alphorm.com

Les bases du langage Boucles et conditions

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

Plan de la vidéo

- Introduction aux structures de contrôle
- Mise en pratique
- À retenir
- Aller plus loin

Introduction aux structures de contrôles

- Permet de contrôler quel code à exécuter en fonction de l'état du programme
- Notion de bloc, finissant toujours par **end**
- Bonne pratique: Indentez votre code! Utilisez "Tab" pour montrer que le code est dans un bloc

Mise en pratique!

Contrôler quelle action demande l'utilisateur

À retenir

- Embranchements vrai/faux

```
if test
  puts "test est vrai"
else #Le block "else" est optionnel
  puts "test est faux"
end
```

- Utilisez `and/or` ou `'&&'/ '||'` pour tester plusieurs conditions en même temps!

À retenir

- Embranchements multiples:

```
case variable
  when 1
    puts "Exécuter lorsque variable vaut 1"
  when 2
    puts "Exécuter lorsque variable vaut 2"
  #when xx...
  else
    puts "Toutes les autres valeurs exécuterons ce code"
end
```

À retenir

- Boucle while:

```
x = 0
while x < 5
  puts "#{x}"
  x+=1
end
```

/!\ Attention aux boucles infinies!

À retenir

- La condition peut être à la fin:

```
x = 0
begin
  puts "#{x}"
  x+=1
end while x < 5
```

- **Note:** Le code sera toujours exécuté une fois minimum!

À retenir

- Boucle for:

```
for x in 0..4
  puts "#{x}"
end
```

- En ruby, nous préférons utiliser l'itération **each**:

```
(0..4).each do |x|
  puts "#{x}"
end
```

Aller plus loin

- Il existe d'autres structures de contrôle en ruby:

```
unless x == if not x
until x == while not x
loop/break
```

- Un listing complet ici:
<http://fr.wikiversity.org/wiki/Ruby/Boucles>

Ce qu'on a couvert

- Les boucles **for**, **while**
- La fonction itérative **each**
- Les conditions **if/else/elsif**
- Les conditions à cas multiples **case/when**



Le langage Ruby

alphorm.com™©



Alphorm.com

Les bases du langage

Les tableaux

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

Plan de la vidéo

- La notion de tableau
- Mise en pratique dans taskman
- À retenir
- Aller plus loin

Les tableaux

- Un tableau est un objet qui contient d'autres éléments et permet de les parcourir,
- En Ruby, de nombreuses fonctions sont présentes pour accéder ou transformer le tableau et ses éléments
- On initialise un tableau très facilement:

```
mon_tableau = []  
#ou  
mon_tableau = Array.new  
#Nous verrons "new" très prochainement
```

À retenir

- **Array#shift** : Retourne le premier élément du tableau et le supprime du tableau
 - Note : **Array#pop** fait la même chose, avec le dernier élément!
- **String#split** découpe en un tableau de chaînes selon un motif délimiteur.

Aller plus loin

- Pour initialiser un tableau, une autre façon d'écrire très lisible :

```
fruits = %w(pomme poire ananas)
# équivalent de
fruits = ["pomme", "poire", "ananas"]
```

Plus rapide non? ☺

Aller plus loin

- Pour les plus connaisseurs, nous aurions pu utiliser des expression régulières:

```
if elm =~ /^flags:/  
  #...
```

- Nous ne verrons pas les expressions régulières dans cette formation, car cela mérite une formation complète!

Pour les curieux et les plus téméraires:

http://www.tutorialspoint.com/ruby/ruby_regular_expressions.htm

Aller plus loin

N'hésitez pas à découvrir l'ensemble des fonctions des tableaux et des chaînes de caractères! Elles vous feront gagner un temps précieux!

<http://ruby-doc.org/core-2.2.0/String.html>

<http://ruby-doc.org/core-2.2.0/Array.html>

Ce qu'on a couvert

- La notion de tableau
- Itération dans les tableaux
- Utilisation de méthodes des chaînes de caractères
- Utilisations de méthodes des tableaux.



Le langage Ruby

alphorm.com™©



Alphorm.com

Les bases du langage

Les hashes

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

Plan de la vidéo

- Notion abordée : Les symboles et les hashes
- Mise en pratique dans *taskman*
- À retenir

Les symboles

- Les symboles sont des mots... Prévus pour être utilisés uniquement dans le code.
- Ils sont définis de cette manière:

```
symbole = :un_symbole
```
- Bizzare? Mais très malin!
 - Le code est beaucoup plus lisible : On scinde ce qui est affichable/affiché (les chaînes) avec ce qui est interne au code! (les symboles)

Les hashes

- Un hash est un objet composé de paires clé/valeur

```
hash = { :cle => "Valeur", :seconde_cle => "Seconde  
valeur" }
```

- Un symbole comme clé prend tout son sens ici!

Vous voyez l'application dans le projet?

Je vous laisse réfléchir quelques secondes...

Les hashes

- La réponse :

- On va utiliser les hashes pour stocker nos tâches!

```
ma_tache = { id: 1, contenu: "Utiliser les hashes", flags: %w(excitant  
important) }
```

- On accède aux éléments du hash comme dans un tableau :

```
ma_tache[:id] #Renvoi 1
```

- L'accès en écriture est elle aussi aisée:

```
ma_tache[:id] = 2 #Modifie le champ id!
```

À retenir

- Les hashes vous permettent de créer et stocker des structures complexes.
- Ils sont très utilisés en étant passé en paramètres nommés dans des fonctions:

Ex:

```
tag :a, href: "http://monlien.com", "cliquez ici"  
#retourne la chaine '<a href='http://monlien.com'>cliquez ici</a>'
```

Ce qu'on a couvert

- Les symboles : ni chaîne, ni chiffres (entier), l'objet parfait pour stocker des valeurs textuelles internes au projet.
- Les hashes : permettent de gérer des clés/valeur. Les valeurs sont accessibles par les clés.
 - Très utile pour stocker des objets avec plusieurs champs.





Alphorm.com

Les bases du langage

Les fonctions

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©



Plan de la vidéo

- Notions abordée : Les fonctions
- Mise en pratique dans *taskman*
- À retenir
- Aller plus loin

Le langage Ruby

alphorm.com™©

Les fonctions

- Notre code est un peu *en vrac* non?
 - Nous allons créer des fonctions pour résoudre ce problème!
- Les fonctions fonctionnent comme des "boîtes noires"
 - N entrée(s), 0 ou 1 sortie
- Créer une fonction :

```
def mafonction parametre1, parametre2
  #Code de la fonction ici
end
```

À retenir

- Une fonction agit comme une boîte noire : Une fois que vous connaissez les entrées et les sorties, vous pouvez modifier le code à l'intérieur sans effet de bord.
- La dernière opération lancée dans le corps de la fonction définira la valeur de retour.
- Vous pouvez retourner d'une fonction à tout moment grâce au mot clé **"return"**

Aller plus loin

- Vous pouvez ajouter des paramètres optionnels dans vos fonctions :

```
# Ce code pourra être appelé avec 1 ou 2 arguments.  
# Si un argument uniquement, alors optionnel prendra comme  
# valeur "Hash vide"  
def ma_fonction obligatoire, optionnel={}  
  
end
```

- Ruby autorise des fonctions à paramètres multiples :

```
# Si j'appelle sonne(1, 2, 3), alors 1,2,3 seront  
# dans le tableau "multiple".  
def somme *nombres  
  sum = 0  
  nombres.each{|x| sum+=x}  
  sum #La dernière ligne lue est la sortie!  
  
end
```

Ce qu'on a couvert

- Nous savons utiliser les fonctions
- Notre code est plus lisible. Nous avons découpé chaque "action" de notre code en différentes fonctions.
- *each, map...* Ces fonctions sont spéciales et demande un bloc de code après...
 - Pouvons nous écrire la même chose?
 - La réponse au prochain épisode!





Alphorm.com

Les bases du langage

Le fonctionnel

Site : <http://www.alphorm.com>
Blog : <http://www.alphorm.com/blog>
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©



Plan de la vidéo

- Rappelez vous le dernier cours...
- La force de *yield*!
- Application dans le projet
- À retenir
- Aller plus loin

Le langage Ruby

alphorm.com™©



Rappelez vous le dernier cours...

```
liste_des_taches.map{|x| x[:id] }.max + 1
```

- Étrange? Non, puissant!
- Ici nous passons un bloc à la fonction `Array#map`
- **Avantage** : Nous pouvons enrober un comportement (ici "parcourir et appliquer une transformation") tout en laissant la liberté d'implémenter ce comportement (ici "retourner l'id uniquement")
- Cette fonctionnalité est utilisée partout en Ruby! Autant l'intégrer dans notre projet (oui mais où?)



La force de yield

- `yield` est un mot du langage qui permet d'activer un bloc passé en paramètre de fonction!
- ```
def faire_si valeur
 if valeur
 yield #On lance le bloc voulu!
 end
end

faire_si 1>0 do
 puts "un est bien supérieur à zero"
end
```

## À retenir

---

- Les fonctions lambda sont des fonctions qui peuvent être stockées comme une variable:
  - `fois2 = lambda{ |x| x*x }`
- Pour appeler la fonction, il suffit d'utiliser la methode call:
  - `fois2.call(4) #retourne 16`
- Vous pouvez passer une fonction lambda comme un bloc:
  - `selecteurTri = lambda{ |a,b| a.prix <=> b.prix }`
  - `produits.sort(&selecteurTri)`
  - Notez "&", qui dit que le paramètre doit être traité comme un bloc de fin de fonction

## Aller plus loin

---

- Vous pouvez vérifier qu'un block a bien été passé en utilisant "block\_given?"

```
def fonction
 yield if block_given?
end
```
- Il existe d'autre manières de traiter un bloc passé qu'en utilisant "yield", la notation &block
- En savoir plus :
  - [http://www.tutorialspoint.com/ruby/ruby\\_blocks.htm](http://www.tutorialspoint.com/ruby/ruby_blocks.htm) (anglais)

## Ce qu'on a couvert

---

- Le passage de bloc aux fonctions
- Le mot clé yield
- La création de block "lambda"



Le langage Ruby

alphorm.com™©



**Alphorm**.com

## Les bases du langage

---

# Les modules

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- Notions abordée: Les modules
- Mise en pratique dans taskman
- À retenir
- Aller plus loin

## Les modules

---

- Les modules permettent de ranger vos fonctions dans des espaces privés permettant de rendre votre code plus lisible.
- Ils permettent aussi les structures appelée "*mixins*".
- Le code de notre application a besoin d'un petit coup de *refactoring* (réécriture)

## Les modules

---

- Déclarer un module:

```
module MonModule
 def self.ma_fonction
 end
end
```

`MonModule.ma_fonction` #ou `MonModule::ma_fonction`

- Notez le mot clé self: Ici il représente "MonModule" et dit "créer une fonction globale dans le module".
- Sans l'usage de self, la fonction ne pourrait être appelée correctement.

## À retenir

---

- @variable
  - Déclare un attribut privé au module (nous ne pouvons accéder à l'attribut hors du module!).
  - C'est donc une variable dans l'espace local du module, très pratique!
    - Nous nous efforçons de supprimer les variables globales utilisées dans le projet!
- Les modules peuvent s'imbriquer. Ex:

```
module Ecosysteme
 module Vegetaux
 def self.planter
 end
 end
end
```

On accède alors aux fonctionnalités ainsi: `Ecosysteme::Vegetaux.planter`

## Aller plus loin

---

- L'usage des modules comme composition de classe (*mixins*) ne sera pas abordé dans ce cours. Cependant n'hésitez pas à rechercher sur Internet!
- Les modules permettent de ranger aussi des classes (nous verrons les classes juste après!)

## Ce qu'on a couvert

---

- Usage des modules pour "ranger" nos fonctions
- Usage du mot clé *self*





# Alphorm.com

## Les bases du langage

# Les classes 1/2

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

Yacine PETITPREZ

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©



## Plan de la vidéo

- Qu'est-ce qu'une classes
- Déclarer une classe
- Les attributs
- Le mot clé "self"
- Application dans le projet
- À retenir

Le langage Ruby

alphorm.com™©

## Qu'est-ce qu'une classe

---

- Une classe permet de lier du code et des variables à un objet
  - Une classe définit la structure d'un objet et les méthodes qui sont applicables dans le contexte de l'objet.
  - Le concept de classe est un peu déroutant pour un débutant, mais très simple à appréhender
    - Exp: Dans la vie de tout les jours, un chien a un nom (attribut) et abois (méthode):

- En Ruby:

```
medor = Chien.new
```

```
medor.name = "Medor" #attribut assigné
```

```
medor.aboyer #woof woof!
```

## Déclarer une classe

---

```
class Chien

 attr_accessor :nom

 def initialize
 @nom = "<sans nom>"
 end

 def aboyer
 puts "#{self.nom} fait woof woof!"
 end

end
```

## Attributs

---

En Ruby, les attributs d'un objet (@attribut) ne sont pas visibles à l'extérieur de la classe!

Afin de pouvoir accéder à l'attribut, nous devons créer des méthodes d'accès (getter/setter):

```
def nom #Getter pour nom
 @nom
end

def nom= x #Setter pour attribut "nom"
 @nom = x
end
```

`attr_accessor` est juste une macro pour effectuer le code ci-dessus.

## Attribut en lecture seule

---

- Les attributs peuvent etres disponible en lecture seule uniquement:

```
attr_reader :nb_aboyement #Nombre de Woof Woof!
```

- **Pour récapituler**: En Ruby vous n'accéder qu'à des méthodes de classe hors de cette classe!

## Le mot clé "self"

---

- Le mot clé self retourne l'objet sur laquelle s'applique la méthode.
- Lorsque nous faisons  

```
medor . aboyer
```
- C'est medor qui aboie, et pas laika!
- **Interdit** : appeler "Chien.aboyer" (car qui aboie dans ce cas?)

## Application dans le projet

---

- C'est gentil les chiens, mais place à la pratique!
- Nous allons créer une classes:

```
class Task
```

- Attributs: contenu, date, flags
- Méthode: afficher, sauvegarder, charger

## À retenir

---

- Le constructeur:

```
def initialize
 #Remplir les champs

end
```

- Cette fonction est lancée automatiquement lors de la création de l'objet (ex: Task.new).

## Ce qu'on a couvert

---

- Créer une classe simple
- Le mot-clé `self`
- Les accesseurs `attr_accessor` / `attr_reader`
- Le constructeur `initialize`





# Alphorm.com

## Les bases du langage

# Les classes 2/2

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

**Le langage Ruby**

**alphorm.com™©**



## Plan de la vidéo

- Héritage
- Application dans le projet
- À retenir
- Aller plus loin

**Le langage Ruby**

**alphorm.com™©**

## Héritage

---

- L'héritage permet de **spécialiser** une classe
- Une classe spécialisée hérite des méthodes et attributs de la classe mère mais peut ajouter/modifier des méthodes et attributs.

## Héritage

---

```
class ChienPolicier < Chien
 attr_accessor :matricule

 def initialize nom, matricule
 super(nom)

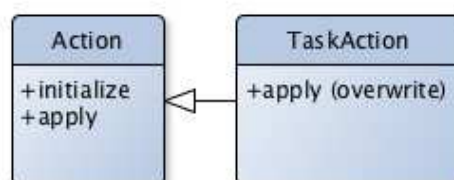
 @matricule = matricule
 end

 def chercher_drogue
 puts "*renifle*"
 end

 #Redéfinition de "aboyer"
 def aboyer
 puts "Un chien policier est bien dressé, il n'aboit pas!"
 end
end
```

## Application dans le projet

---



## Application dans le projet

---

C'est parti! Let's do it! 😊

## À retenir

---

- Notre programme est plus aisé à maintenir. Nous pouvons ajouter de nouvelles actions facilement maintenant!
- Nous nous assurons de ne pas nous répéter
- En spécialisant nos actions, nous écrivons une seule fois le code de spécialisation (aller chercher la tâche).

## Aller plus loin

---

- Nous avons effleuré le concept de classes. Vous trouverez une dense documentation sur internet!
- Les mots clés **private/public/protected** permettent de rendre les fonctions à l'usage interne ou externe. C'est une bonne pratique de les utiliser!
- Les classes souvent suivent des "patrons de conception" (design templates) ou "bonnes pratiques d'architecture".

## Ce qu'on a couvert

---

- Fin du chapitre 2!
  - Nous avons vu les grandes lignes et structures du langage Ruby!
  - Vous êtes capable de faire un petit programme
- Cependant, notre programme a toujours un **gros** problème:
  - **On teste, mais on ne sauvegarde/charge pas nos taches!**
- Prochain chapitre: Ruby avancé
  - (et puis, on va finalement sauvegarder notre liste de taches!)



Le langage Ruby

alphorm.com™©



**Alphorm**.com

## Ruby avancé

---

## Installer et utiliser les gems

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**  
Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- Qu'est-ce qu'une gem
- Les fichiers JSON
- Application dans le projet
- À retenir
- Aller plus loin

## Qu'est-ce qu'une gem

---

- Bibliothèque de code tierce.
- Ajoute des méthodes, objets et fonctions dans votre programme
- Vous permet de ne pas réinventer un concept déjà existant (où de nombreuses personnes ont travaillé longtemps pour vous fournir le meilleur).
- Installer une gem :

```
gem install <nom de la gem>
```

## Les fichiers JSON

---

- JSON = JavaScript Object Notation

- Permet de sauvegarder et charger un hash depuis une chaîne de caractère

`require 'json'`

```
hash = { example: "voici un exemple" }
```

```
str = hash.to_json #Renvoi une chaîne de caractère avec les données du hash.
```

```
new_hash = JSON.parse(str) #Retourne notre hash
```

Note: Le nouveau hash retourné a comme clé des strings! (pas des symboles!)

```
{ "example" => "voici un exemple" }
```

## Aller plus loin

---

- YAML: Autre format de stockage de données (surement plus utilisé dans l'environnement Ruby)
- Explorer les gems sur [Rubygems.org](http://Rubygems.org) / [github](https://github.com)!

## Ce qu'on a couvert

---

- Installer une gem
- Charger la gem via **require**
- Charger et sauvegarder les fichiers via `File.read` / `File.open`
- Utiliser le format JSON pour sauvegarder un Hash de données



Le langage Ruby

alphorm.com™©



**Alphorm**.com

Ruby avancé

---

Éclater le code dans  
plusieurs fichiers.

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- Découper votre code en fichiers
- Mise en pratique
- Ce qu'on a couvert

## Découper votre code en fichiers

---

- Lorsque le code devient plus gros, il est préférable de gérer plusieurs fichiers.
- On colle notre code dans chaque fichier en fonction de leur contexte (ex: 1 fichier par module, 1 fichier par classe...)
- Si charger une gem est facile, charger les fichiers Ruby n'est pas forcément une sinécure (on va voir ça!)

## Ce qu'on a couvert

---

- Utiliser **require** avec un chemin = charger un fichier (et non une gem).
- Utilisation de la constante **\_\_FILE\_\_** pour récupérer le chemin du fichier exécutant.
- Notre code est maintenant plus propre et ressemble (enfin!) à un vrai projet!



Le langage Ruby

alphorm.com™©



**Alphorm**.com

## Ruby avancé

---

# Utilisation du Bundler

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**  
Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- Qu'est-ce que le bundler
- Utiliser le Gemfile
- Application dans le projet
- À retenir
- Aller plus loin

## Qu'est-ce que le Bundler

---

- Permet de gérer la liste et les versions de chaque gem utilisée par votre projet
- Permet d'assurer une compatibilité si votre collègue souhaite travailler sur votre projet
- Permet d'installer rapidement toutes les gems nécessaires au bon fonctionnement de votre projet sans devoir le faire manuellement une par une.

## Utiliser le Gemfile

---

- Le Gemfile centralise les informations à propos des gems utilisées par votre application.
- Bundler se base sur ce fichier pour télécharger et installer les gems nécessaires au bon fonctionnement.
- Taper "bundle install" pour installer les dépendances manquantes.

## À retenir

---

- Le bundler permet de simplifier la gestion des dépendances.
- Il permet à vos utilisateurs d'installer toute les dépendances pour lancer votre projet.
- Il simplifie l'usage des "require" dans votre projet.
- Que du bon! 😊

## Aller plus loin

---

- Tout Ruby utilise bundler! Recherchez une gem sur internet, et simplement ajoutez là dans votre Gemfile! Aussi simple que ça!
- Bundler permet aussi de créer des gems (nous ne verrons pas cette partie dans notre cours). Une gem peut être utilisée pour déployer votre application très facilement (il s'agit d'un "binaire" Ruby!).

## Ce qu'on a couvert

---

- Créer un Gemfile
- Utiliser le bundler pour charger toutes nos dépendances d'un coup.



**Alphorm**.com

## Ruby avancé

---

# La réflexion du langage

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

**Le langage Ruby****alphorm.com™©**

## Plan de la vidéo

---

- Qu'est-ce que la réflexion?
- Liste des fonctionnalités intéressantes
- Application dans le projet
- À retenir

**Le langage Ruby****alphorm.com™©**

## Qu'est-ce que la réflexion?

---

- La réflexion (reflection en anglais) c'est la possibilité pour le langage de lister les variables, méthodes, nom de classes.
- Le langage peut alors agir sur lui-même
- On peut automatiser des créations de fonctions / classes, appeler des méthodes grâce à un paramètre (et non plus le nom de la méthode directement)...
- C'est un outil puissant qui est largement utilisé en Ruby (ex: *ActiveModel*, *ORM de Ruby On Rails*).

## Liste des fonctionnalités intéressantes

---

- **methods** liste toute les méthodes de l'objet.
- **send** appelle une méthode.
- **respond\_to?** vérifie qu'une méthode existe.
- **is\_a?** Vérifie qu'un objet hérite bien d'une classe
- **class** retourne la classe de l'objet en cours
- et bien d'autres choses...!

## Ce qu'on a couvert

---

- Initiation aux méthodes de réflexion.
- Utilisation dans le projet pour automatiser la création de champs.



Le langage Ruby

alphorm.com™©



**Alphorm**.com

## Ruby avancé

---

## La structure DSL

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- Qu'est-ce que les DSL?
- Créer un DSL
- Application dans le projet
- À retenir

## Qu'est-ce que les DSL?

---

- Les DSL sont des structures de code très populaire en Ruby On Rails
- Elles permettent de rendre le code de définition claire et concis
- Elles demandent un peu de travail pour mettre en place, au profit d'une utilisation plus simple et rapide
- Elles reposent sur des fonctionnalités avancées du Ruby.



## Créer un DSL

---

- Sans DSL:

```
action = Action.new :list_done do
 #faire l'action
end
action.description = "Liste les taches réalisées"
ActionManager.add action
```

- Avec DSL:

```
ActionManager.register do
 description "Liste les taches réalisées"
 action :list_done do
 #faire l'action
 end
end
```



## À retenir

---

- Utilisation de *instance\_eval* pour lancer un bloc dans le contexte d'un objet
- Les DSL nécessitent de créer une classe qui active les fonctionnalités du DSL
  - C'est assez long à mettre en place.
  - Mais c'est beau et puissant. Gain de temps si nous devons définir une centaine d'actions par exemple!
  - Ces structures sont très utilisées (RoR), maintenant vous connaissez la magie derrière cela!

## Ce qu'on a couvert

---

- Qu'est-ce qu'un DSL
- Créer une structure DSL
- Nous avons démystifié le code Ruby



Le langage Ruby

alphorm.com™©



**Alphorm**.com

## Ruby avancé

---

## Le monkey-patching

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- Réouverture de classes
- Qu'est-ce que le *monkey patching*?
- Application dans le projet

## Réouverture de classes

---

- En Ruby, la redéfinition d'une classe existante ne l'écrase pas. Attributs et méthodes nouvelles sont ajoutés à la classe existante.
- Si les attributs et méthodes existent déjà, elles sont réécrites (écrasement).
- Permet:
  - Scinder le code d'une classe complexe dans plusieurs fichiers
  - Réécrire un comportement *à posteriori*
  - Améliorer/réadapter un code qui n'est pas de vous

## Monkey Patching

---

- Le *monkey patching* définit le fait de réécrire le comportement d'une classe qui n'est pas la votre!
  - Exemple:
    - Ajouter des méthodes aux classes de base (String, Array...)
    - Modifier le comportement d'une classe qui est utilisée par plusieurs gems.
- **Avantages** 😊
  - Clarté du code écrit à l'aide de ces nouvelles méthodes
    - Date.yesterday => Retourne la date d'hier
- **Inconvénients** ☹️
  - Votre modification s'étend à toute les gems qui utilisent cette classe
  - Risque de collision avec les ressources utilisant cette méthode.

Le langage Ruby

alphorm.com™©

## Ce qu'on a couvert

---

- Nous connaissons la notion de Monkey Patching
- Nous avons démystifié le concept!
- Nous avons ajouté un comportement à la classe Object, afin de pouvoir lancer des fonctions sans tester si l'objet est vide (nil).
- A noter que cette fonctionnalité existe dans Ruby On Rails par défaut!



Le langage Ruby

alphorm.com™©



# Alphorm.com

## Ruby Avancé

### Finaliser le projet & gérer les erreurs

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**

Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

**Le langage Ruby**

**alphorm.com™©**

## Plan de la vidéo

- Gérer les erreurs en Ruby
- Application dans le projet
- À retenir

**Le langage Ruby**

**alphorm.com™©**

## Gérer les erreurs en Ruby

---

- Le mot clé `rescue` "récupère l'erreur"

```
begin
 faire_une_operation
rescue => e
 puts "Erreur: #{e}"
end
```

## Gérer les erreurs en Ruby

---

- Utiliser `raise` pour déclencher une erreur

```
if x.nil?
 raise "x ne doit pas etre nul"
end
```

Vous pouvez aussi passer une classe héritant de `RuntimeError`.

## Aller plus loin

---

- Gérer différentes erreurs:

```
begin
 faire_une_operation
rescue SpecialError => e

 puts "Gérer cette erreur de manière spéciale"
rescue => e
 puts "Erreur: #{e}"
end
```

Note: Les spécialisations sont toujours écrites en premier

## Aller plus loin

---

- S'assurer que du code s'exécute quoi qu'il arrive:

```
f = File.open("monfichier", "wb")
begin
 #faire des operations sur le fichier
ensure
 f.close
end
```

## Ce qu'on a couvert

---

- Finalisation du projet...
  - À vous d'implémenter les grandes fonctionnalités à l'aide de vos nouveaux acquis!
- Initiation à la gestion des erreurs!



Le langage Ruby

alphorm.com™©



**Alphorm**.com

## Conclusion du cours

---

## Le langage **Ruby**

Site : <http://www.alphorm.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Yacine PETITPREZ**  
Développeur Fullstack Ruby On Rails  
Gérant Red Tonic Solutions

Le langage Ruby

alphorm.com™©

## Plan de la vidéo

---

- En résumé
- S'entraîner
- Aller plus loin
  - Sinatra
  - Ruby On Rails

## En résumé

---

- Nous savons lire et écrire du code Ruby!
- Nous comprenons que les gems peuvent altérer les fonctionnalités existantes du langage
- Nous comprenons que l'écriture "magique" du Ruby n'est en fait pas magique du tout!
- Nous sommes prêt pour commencer à travailler avec un framework Web!

## S'entrainer

---

- Vous pouvez finir ce projet! Des éléments et fonctionnalités n'ont pas été encore implémentée! À vous de jouer!
  - Récupérer les sources du projet ici: <https://github.com/redtonic/alphorm-ruby>
- Vous pouvez tenter de "packager" le projet dans une gem! Cela vous permettra d'appeler le binaire via "taskman" et non plus "./taskman"
  - De plus, vous pourrez partager votre petite application avec vos amis!
- Vous pouvez commencer à voir plus gros, et orienté Web!

## Aller plus loin

---

- Sinatra + ERB:
  - **Sinatra**: Serveur web léger pour créer de petites applications Web.
  - **ERB**: EMBEDDED RUBY. Permet de créer des templates de pages web (html) avec du code Ruby dedans.
  - Apprendre à coder un serveur Web facilement en Ruby. Bonne initiation avant Ruby on Rails.
  - <http://code.tutsplus.com/tutorials/singing-with-sinatra--net-18965>

## Aller plus loin

---

- Ruby On Rails:
  - Créer une application Web plus complexe avec le modèle MVC de Ruby On Rails.
  - Rumeur: Un cours serait à venir sur Alphorm spécialisé sur Ruby On Rails! ☺

## Fin

---

- Fin de cette initiation à Ruby!
  - Des questions? Des remarques?
    - [yacine@redtonic.net](mailto:yacine@redtonic.net)
    - Les commentaires Alphorms!
    - Forum d'Alphorm

