

The background features a complex, abstract composition of overlapping red and orange-red geometric shapes. These shapes include large, rounded polygons and faceted, crystalline forms that create a sense of depth and movement. The colors vary in intensity, from deep, dark reds to lighter, more vibrant oranges, with soft gradients and highlights that suggest a three-dimensional, polished surface. The overall effect is dynamic and modern.

**Chargement en cours**

# Présentation de



# Ruby

Rencontres Mondiales du Logiciel Libre 2007  
Amiens

# Avant-propos :: présentation de l'orateur

Guillaume DESRAT / Zifro AKA guillaumed  
<[guillaume.desrat@rubyfr.org](mailto:guillaume.desrat@rubyfr.org)> & <http://zlab.fr/>

- Rubyiste depuis 2002, Railer depuis 2005



- Secrétaire de l'association
- Participe aux listes de diffusion RubyFR et RailsFrance
- Répond à vos questions sur IRC
- Développeur informatique chez Cersi, au Luxembourg  
(<http://www.cersi.com>)



# Présentation de



# Ruby

Présentation

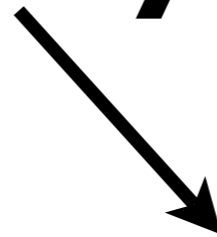
On pourrait parler des heures de Ruby (ne serait-ce qu'au sujet de la méta-programmation...).

Le but : vous donner un aperçu du langage Ruby, vous donner les clés pour comprendre les conférences suivantes.

# Présentation de



# Ruby



Ruby

Pas Ruby vs. Python.

Pas Ruby vs. JAVA.

Pas Ruby on Rails.

Merci.

# Le langage :: historique

- créé en 1993 par un japonais : Yukihiro Matsumoto
- publié pour la première fois en 1995
- emprunte les excellentes idées de nombreux autres langages : Perl, Python, Smalltalk, Ada, Lisp, Eiffel, ...
- le but avoué : faire de Ruby un langage naturel

## JRuby (<http://jruby.codehaus.org>)

- une implémentation du langage Ruby en JAVA
- permet d'utiliser le meilleur des deux langages
  - dériver une classe JAVA en Ruby
  - dériver une classe Ruby en JAVA

## Ruby CLR (<http://rubyforge.org/projects/rubyclr/>)

- une implémentation du langage Ruby pour la plateforme .NET

Ruby est :

- un langage de programmation
- un langage objet
- un langage lisible
- un langage dynamique

Ruby est ...

**un langage de programmation**

# Un langage de programmation :: syntaxe des identificateurs

- `une_variable_locale`
- `un_nom_de_methode`
  
- `@une_variable_d_instance`
- `@@une_variable_de_classe`
  
- `$une_variable_globale, $UNE_AUTRE`
  
- `UneConstante, UNE_AUTRE_CONSTANTE`
  
- `:un_symbole`

# Un langage de programmation :: conventions

- les noms de classe sont des ConstantesEnCamelCase (ex : String, TimeZone)
- les autres constantes sont TOUT\_EN\_MAJUSCULES (ex : RUBY\_VERSION, ARGV)
- les noms des méthodes peuvent se terminer par ! ou ?
  - “!” pour les méthodes qui modifient l’objet (ex : chaine.strip!)
  - “?” pour les méthodes qui renvoient un booléen (ex : obj.nil?)

# Un langage de programmation :: structures de contrôle

- conditions :
  - if ... (then) ... else ... end
  - unless ... (then) ... else ... end
- mécanisme d'exception :  
begin ... rescue Exception ... ensure ... end
- blocs :
  - do ... end
  - { ... }

Ruby est ...

un langage objet

# Un langage objet :: (presque) tout est objet

Dans Ruby, le paradigme objet est très poussé ; exemple :

<code>-123.abs</code>	<code># renvoie 123, car -123 est une</code> <code># instance de la classe Fixnum</code>
<code>'rml'.upcase</code>	<code># renvoie 'RMLL' car 'rml' est une instance</code> <code># de la classe String</code>

Il n'y a pas de type de base, que des classes.

# Un langage objet :: déclaration d'une classe :: les bases

## Déclaration d'une classe "Presentation" :

```
class Presentation          # le nom de la classe

  @@nombre = 0            # une variable de classe, pour compter le nombre de présentation

  def initialize (titre, duree=15) # déclaration de la méthode initialize
    @titre = titre        # qui affecte les paramètres aux variables d'instances de même nom
    @duree = duree        # si duree n'avait pas été passé, il vaudrait 15
    @@nombre += 1        # incrémente le nombre de présentations (++ n'existe pas en Ruby)
  end

  def to_s                 # quand on écrit "puts object", c'est cette méthode qui
    @titre + ':' + @duree.to_s + ' minutes' # renvoie ce qui doit être affiché
  end
end                        # pas besoin de return, Ruby renvoie à l'appelant la
                          # dernière expression évaluée
```

# Un langage objet :: instantiation d'une classe

## Instantiation de la classe "Presentation" :

```
p = Presentation.new('Introduction à Ruby', 45)
puts p # 'Introduction à Ruby : 45 minutes'
```

Les méthodes de cet objet peuvent être appelées via la syntaxe `<objet>.<methode>`

# Un langage objet :: les modules

Les modules ont deux utilisations :

- espace de nommage (RMLL::EDITION)
- étendre les fonctionnalités d'une classe, ou d'un objet (pseudo héritage multiple)

```
module RMLL
  EDITION = 2007
  def salle
    # ...
  end
end
```

```
class Presentation
  include RMLL
end
```

```
p.extends(RMLL)
```

Ruby est ...

**un langage lisible**

# Un langage lisible :: les blocs

Les blocs (`do ... end` et `{...}`) sont des objets (classe `Proc`).

On peut passer des objets aux méthodes.

On peut donc passer des blocs aux méthodes.

# Un langage lisible :: les blocs

La méthode `yield` exécute le bloc passé en argument (bloc qui peut, lui-même, prendre des paramètres) :

```
class Personne
```

```
  def initialize(nom)
    @nom = nom
  end
```

```
  def salue
    yield(@nom)
  end
```

```
end
```

```
thomas = Personne.new('thomas')
thomas.salue { |n| puts "salut #{n}" }
```

```
# la méthode salue se contente d'exécuter
# le bloc qui lui est passé, avec l'argument @nom
```

```
# affiche "salut thomas"
```

# Un langage lisible :: les blocs

## Utilisations courantes des blocs :

```
f = File::open(mon_fichier)
if f <> -1
  # code
  f.close
end
```

```
File::open(mon_fichier) do |f|
  # code
end
```

```
['Frédéric', 'Alexis'].each do |lyonnais|
  lyonnais.rate_son_train
end
```

# lire et manipuler le contenu d'un fichier

# le fichier est automatiquement fermé

# avec un itérateur

# pour exécuter un même traitement sur une

# collection d'objet

# Un langage lisible :: les itérateurs

- des méthodes (éventuellement adjointes par Enumerable)
- un moyen d'améliorer la lisibilité du code
- une façon naturelle d'écrire un programme
- les compagnons idéaux des blocs

# Un langage lisible :: les itérateurs

“pour chacun des éléments de la collection, exécute le code du bloc passé en paramètre, en lui passant l’objet courant de la collection”

```
5.times { puts “Bienvenue à Amiens” }  
5.times { |i| puts i }
```

```
# affiche 5 fois le message  
# affiche 1, 2, 3, 4 et 5
```

```
[‘Frédéric’, ‘Alexis’].each do |lyonnais|  
  lyonnais.rate_son_train  
end
```

```
# avec un itérateur  
# pour exécuter un même traitement sur une  
# collection d’objet
```

Ruby est ...

un langage dynamique

# Un langage dynamique :: introspection

Les composants du langage ne sont pas verrouillés :

- ajout de méthodes
- redéfinition
- suppression

```
class String
  def ma_methode_utile
    # code
  end
end
```

```
# on "rouvre" la définition de la classe String
# on déclare une nouvelle méthode

# cette méthode est maintenant disponible pour tous
# les objets de la classe String
```

# Un langage dynamique :: méthode singleton

Les objets créés peuvent être rendus spécifiques :

- ajout de méthodes
- redéfinition
- suppression

```
p = Personne.new('Thomas')
def p.affiche
  puts "Je suis #{@nom}"
end
p.affiche
```

```
# on crée une instance de notre classe Personne
# on déclare une méthode affiche pour cet objet

# affiche "Je suis Thomas"
```

# Un langage dynamique :: méta-programmation

Le langage offre des méthodes pour évaluer du code dynamiquement :

- `eval`
- `instance_eval`
- `class_eval`
- `module_eval`

Permet d'interpréter du texte comme du code, pendant l'exécution

# Un langage dynamique :: méta-programmation

Le développeur peut utiliser des 'hooks' pour intercepter :

- ajouts de méthodes
- inclusion d'un module dans une classe
- appel à une méthode inexistante
  
- et plus encore

Création “automagique” de méthodes dynamiques (très utilisée dans Ruby on Rails) :

- `method_missing` intercepte l'appel à une méthode inexistante
- le développeur détermine s'il doit la traiter
  - exception
  - appel d'une autre méthode, pas d'exception levée

Quel est l'avenir de Ruby ?

Quels outils pour développer ?

Quelles applications existent déjà ?

Quelles références consulter ?

- Aujourd'hui en version 1.8.6
- Une nouvelle version chaque Noël
- Ruby 2.0 (nom de code YARV) pour Noël 2007
  - une réécriture complète de la VM
  - plus rapide

La quasi-totalité des éditeurs offre un support du langage (coloration syntaxique, vérification du code) :

- (g)vi(m)
- (X)emacs
- SciTE
- FreeRIDE
- Eclipse : plugin Ruby Development Tools
- NetBeans 6.0 : support Ruby, JRuby, Ruby on Rails
- TextMate (sous Mac OS X)

`irb`

- Ruby on Rails (<http://www.rubyonrails.org/>)
- MetaSploit (<http://www.metasploit.org/>)
- Nitro / Og (<http://www.nitroproject.org/>)
- Hackety Hack (<http://www.hacketyhack.net/>)

**Est-ce bien important ?**

## Ressources :: sur Internet :: en Anglais

- Le site officiel : <http://www.ruby-lang.org/>
- Le Ruby Application Archive : <http://raa.ruby-lang.org/>
- La forge : <http://www.rubyforge.org/>
- Les listes de diffusion (ruby-talk@, ruby-core@, ...)
- Le groupe comp.lang.ruby sur Usenet
- Le salon IRC #ruby-lang @ irc.freenode.net
- Les blogs (<http://www.whytheluckystiff.net/> est à visiter)

## Ressources :: sur Internet :: en Français

- Le site de l'association Ruby France :  
<http://www.rubyfrance.org/> <http://www.rubyfr.org/>  
(disponibles aux deux adresses)
- Le site officiel, traduit : <http://www.ruby-lang.org/>
- Les listes de diffusion sur Google Groups
- Le groupe fr.comp.lang.ruby sur Usenet
- Le salon IRC #rubyfr @ irc.freenode.net
  
- Les blogs des utilisateurs francophones bien sûr !

- Ruby In a Nutshell
- Programming Ruby (le fameux “Pickaxe”)
- The Ruby Way
- Ruby par l’exemple

Passez sur le stand de l’association Ruby France, il y a des ouvrages sur le sujet.

# Des questions ?

Merci