Εισαγωγή στη Ruby

Εισαγωγή

Η Ruby σχεδιάστηκε από τον Yukihiro "Matz" Matsumoto στην Ιαπωνία περίπου το 1995. Σκοπός του Yukihiro ήταν η δημιουργία μίας γλώσσας προγραμματισμού που θα διευκόλυνε τον προγραμματιστή να επικεντρώνεται κυρίως στη συγγραφή κώδικα και όχι στην επίλυση παράπλευρων προβλημάτων που συχνά προέκυπταν από την ίδια τη φύση της γλώσσας. Έτσι δανείστηκε τα δυνατά γνωρίσματα άλλων γλωσσών όπως π.χ. Python, Perl, Smalltalk, Eiffel και Lisp. Πλέον η γλώσσα εξελίχτηκε αρκετά και έχει κερδίσει αρκετούς προγραμματιστές. Αυτή τη στιγμή βρίσκεται στην έκδοση 1.9.3.

Η επίσημη ιστοσελίδα είναι η http://www.ruby-lang.org/en/, όπου μπορείτε να δείτε και προτεινόμενη βιβλιογραφία. Επίσης καλό είναι να επισκεφτείται τη http://tryruby.org/levels/1/challenges/0 όπου, χωρίς να εγκαταστήσετε τοπικά τη Ruby, μπορείται να τρέξετε online κάποια παραδείγματα μέσω irb (Interactive Ruby). Η irb δεν είναι τίποτα άλλο παρά μία διαδραστική , όπως λέει και το όνομά της, κονσόλα της Ruby. Σύνηθης τακτική είναι εκτός από τον αγαπημένο text editor να έχουμε ανοιχτεί και την irb (εγκαθίσταται μαζί με τη Ruby) για γρήγορα πειράματα. Τα ίδια ακριβώς αποτελέσματα θα παίρναμε και αν είχαμε σώσει τον κώδικά μας σε ένα αρχείο με κατάληξη .rb ή .rbw και εκτελούσαμε το αρχείο (προτιμήστε το αρχείο να είναι σε utf8-ρυθμίστε το μέσω του text editor που χρησιμοποιείτε-).

Γενικά χαρακτηριστικά

Το πρώτο που ακούει κανείς στην πρώτη επαφή με τη γλώσσα είναι ότι στη Ruby τα πάντα είναι αντικείμενα. Και αυτό είναι αλήθεια. Ακόμα και το κύριο πρόγραμμα της Ruby (η main θα λέγαμε) είναι ένα αντικείμενο. Σε αυτό το γεγονός έγκειται κατά ένα μεγάλο βαθμό η «δύναμη» της Ruby.

Ακόμη, η Ruby επιρεασμένη από τη small talk προσφέρεται για τη συγγραφή συμπυκνωμένου κώδικα. Αυτό θα φανεί παρακάτω στα παραδείγματα που ακολουθούν.

Επίσης η Ruby είναι cross-platform (Linux, Mac OS, Windows). Δε χρειάζεται κάποιο σύμβολο στο τέλος των εντολών, ούτε εξαρτάται από των αριθμό των κενών όπως π.χ. η Python. Η στοίχιση του κώδικα γίνεται με τον κλασικό τρόπο καθαρά για θέμα αναγνωσιμότητας. Εδώ η Ruby έχει ένα ακόμα δυνατό σημείο. Οι συναρτήσεις που δεν δέχονται ορίσματα μπορούν να κληθούν και χωρίς τις παρενθέσεις π.χ. δοκιμάστε στη σελίδα http://tryruby.org/levels/1/challenges/0 την εντολή «1.class», χωρίς τα εισαγωγικά και την «1.class()»

Ναι, όπως είπαμε τα πάντα είναι κλάσεις. Οπότε ακόμα και οι σταθεροί αριθμοί είναι αντικείμενα. Όπως βλέπουμε η τελεία μας επιτρέπει να καλέσουμε μεθόδους όπως και σε άλλες γλώσσες. Η μέθοδος class() ή class μας επιστρέφει τον τύπο της

κλάσης.

Στην irb λοιπόν θα δείτε το:

```
Interactive ruby ready.
> 1.class
=> Fixnum
> 1.class()
=> Fixnum
```

Με το σύμβολο => συμβολίζεται η έξοδος της κονσόλας. Οι σταθεροί αριθμοί, λοιπόν, όπως το 1 είναι αντικείμενο τύπου Fixnum. Προσέξτε ότι είτε γράψουμε είτε παραλείψουμε τις παρενθέσεις το αποτέλεσμα είναι ίδιο. ΠΡΟΣΟΧΗ! Οι παρενθέσεις μπορούν να παραληφθούν ΜΟΝΟ για τις μεθόδους που δε δέχονται όρισμα. Αυτή η απαλοιφή των παρενθέσεων οδηγεί σε κώδικα με λιγότερα σύμβολα και περισσότερα γράμματα, με αποτέλεσμα ο κώδικας να μοιάζει πιο πολύ με κείμενο παρά με κώδικα. Αυτό βοηθά ώστε ο κώδικας να είναι γενικά πιο ευανάγνωστος και ευκολονόητος, ακόμα και σε περιπτώσεις που πρόκειται για κώδικα τρίτου ατόμου.

Εάν δοκιμάσετε

```
> (2.0).clas s
=> Float
```

Άρα η Ruby τους δεκαδικούς τους έχει ως αντικείμενα τύπου Float.

Ο pointer this που συναντάμε σε άλλες γλώσσες π.χ. C++,C#, Java, PHP κτλ. στη Ruby γίνεται self. Δοκιμάστε το :

```
> self.class
=> Object
```

Εδώ , λοιπόν, κρύβεται αυτό που προαναφέραμε. Ότι ακόμα και όταν ξεκινάμε το πρόγραμμα (απλά ανοίξαμε την irb) βρισκόμαστε μέσα σε ένα αντικείμενο τύπου Object!

Με την ίδια λογική, στη Ruby δεν υπάρχει η τιμή null που συναντάμαι σε πολλές γλώσσες, αλλά το αντικείμενο nil. Δοκιμάστε το εξής:

```
> nil.class
=> NilClass
```

Το αντικείμενο αυτό, λοιπόν, είναι τύπου NilClass.

Δοκιμάστε τώρα το :

```
NilClass.m ethods
=> [:allocate, :superclass, :const_get, :public_instance_methods, :autoload?, :freeze, :const_
missing, :included_modules, :==, :class_exec, :instance_method, :public_class_method, :anc
estors, :<, :protected_method_defined?, :>, :===, :private_instance_methods, :hash, :class_
variables, :<=, :method_defined?, :instance_methods, :class_variable_defined?, :name, :priva
te_method_defined?, :const_set, :autoload, :include?, :protected_instance_methods, :module
_eval, :module_exec, :<=>, :private_class_method, :constants, :to_s, :public_method_define
d?, :class_eval, :>=, :const_defined?, :puts, :methods, :nil?, :=~, :protected_methods, :inspec
t, :tainted?, :is_a?, :instance_variable_get, :instance_variable_defined?, :frozen?, :respond_to
?, :instance_variable_set, :untaint, :respond_to_missing?, :clone, :send, :private_methods, :sin
gleton_methods, :eql?, :type, :dup, :kind_of?, :instance_of?, :taint, :class, :instance_variables,
 :public_methods, :instance_exec, :__send__, :instance_eval, :equal?, :!, :!=]
```

Η συνάρτηση methods θα σας φανεί αρκετά χρήσιμη καθώς σας επιστρέφει όλες τις μεθόδους μίας κλάσης και μπορείτε να το χρησιμοποιήσετε ως πρόχειρο και γρήγορο ευρετήριο. Δοκιμάστε να βρείτε τις μεθόδους του nil με την εντολή nil.methods. Μην τρομάξετε. Δεν κάνατε κάτι λάθος. Απλά για να εξηγήσουμε το γιατί συμβαίνει αυτό θα πρέπει να εξηγήσουμε και κάποια άλλα πραγματάκια οπότε προς το παρόν ας το αφήσουμε στην άκρη.

Τέλος, τα σχόλια μίας γραμμής στη Ruby ξεκινούν μετο σύμβολο #.

Κατηγορίες μεταβλητών στη Ruby

Καταρχήν, στη Ruby δε χρειάζεται να δηλώσετε τύπο μεταβλητής. Θα δείτε όμως κάποια σύμβολα όπως $ και @ οπότε καλό είναι να γνωρίζετε τι σημαίνουν.

Κοιτάξτε τις 3 παρακάτω εντολές (έχετε υπόψη ότι οι μεταβλητές στη Ruby ξεκινούν με πεζό γράμμα και οι κλάσεις με κεφαλαίο):

```
max = 3
$max = 3
@@max = 3
@max
```

Στην πρώτη περίπτωση, η μεταβλητή «max» είναι τοπική. Η εμβέλειά της αφορά το τμήμα κώδικα στο οποίο είναι εμφωλιασμένη. Όπως π.χ. έχουμε τις τοπικές μεταβλητές μέσα σε μπλοκ κώδικα σε άλλες γλώσσες προγραμματισμού.

Η μεταβλητή $max είναι global, δηλαδή ορατή από παντού μέσα στο πρόγραμμα. Γενικά global μεταβλητές στη Ruby δε συνηθίζοντα και θα χρησιμοποιηθούν για τις μεταβλητές που εκφράζουν κάτι που είναι πράγματι global.

Τέλος, η μεταβλητή «@@max» δηλώνει ιδιότητα κλάσης. Έτσι δε θα τη δείτε ποτέ μόνη της αλλά μέσα στη δήλωση μίας κλάσης. Δηλαδή:

```
def Maxclass
        @@max = 3
end
```

Η πρώτη και τρίτη εντολή ορίζουν μία κλάση με το όνομα «Maxclass». Η «@max» είναι ιδιότητα,μεταβλητή δηλαδή της Myclass.

Τέλος η «@max» είναι instance variable. Εάν για παράδειγμα έχουμε 2 αντικείμενα της ίδιας κλάσης:

```
def Maxclass
        @max = 3
end
```

και η τιμή της @max αλλάξει σε ένα τότε στο άλλο δεν αλλάζει. Δηλαδή οι μεταβλητές που ξεκινούν με ένα @ αφορούν μία και μόνο instance της κλάσης για αυτό και λέγονται instance variables. Μπορείτε να τη φανταστείτε ως το ανάποδο της static της Java.Περισσότερα για αυτό θα πούμε στο επόμενο μάθημα όπου θα μιλήσουμε και για κλάσεις.

Τελεστές της Ruby

Στη Ruby θα βρείτε τους γνωστούς τελεστές +-*/ φυσικά. Έχετε υπόψη ότι όταν

κάνετε πράξεις μεταξύ αριθμών στη Ruby δε χρειάζεται να κάνετε typecasting , καθώς αυτό το αναλαμβάνει η ίδια η γλώσσα. Π.χ. στη C/C++ το αποτέλεσμα της διαίρεσης ακεραίων πρέπει να αποθηκευτεί σε αριθμό double ή float. Στη Ruby η αλλαγή γίνεται από τη μηχανή της ίδιας της γλώσσας.

Στη Ruby δε θα βρείτε τα ++ και – αλλά τα +=, -=, *=,/=. Αυτό οφείλεται στο ότι στη Ruby όλα είναι αντικείμενα, οπότε τα σύμβολα +,- [] κτλ. είναι μέθοδοι. Επομένως, η γραφή ++ και – δε συμβαδίζει με αυτή τη λογική.

"Εξαίρεση", θα μπορούσε κανείς να θεωρήσει, είναι η ύψωση στη δύναμη που συμβολίζεται ως **. Βέβαια, σε αυτή την περίπτωση όμως το ** είναι πάλι μία μέθοδος που αναλαμβάνει την ύψωση σε δύναμη, άρα σε τελική ανάλυση δεν είναι πραγματικά εξαίρεση.

Τελειώνοντας με τους τελεστές να πούμε ότι όπως και σε άλλες γλώσσες προγραμματισμού έτσι και εδώ οι τελεστές μπορούν να γίνουν overload. Για παράδειγμα, το σύμβολο της αφαίρεσης μπορεί να χρησιμοποιηθεί σε πίνακες. Έτσι π.χ. δοκιμάστε τον κώδικα που προτείνεται και στην επίσημη σελίδα της Ruby ως παράδειγμα (αρχείο subtrack_arrays.rb):

```
# Ruby knows what you
# mean, even if you
# want to do math on
# an entire Array
cities  = %w[ London
              Oslo
              Paris
              Amsterdam
              Berlin ]
visited = %w[Berlin Oslo]

puts "I still need " +
   "to visit the " +
   "following cities:",
   cities - visited
```

Θα δείτε ως αποτέλεσμα:

```
I still need to visit the following cities:
London
Paris
Amsterdam
```

Όπως βλέπετε έγινε αφαίρεση πινάκων γιατί για την κλάση Array έχει γίνει overload η μέθοδος -. Είπαμε ότι δεν είναι απλά σύμβολο αλλά στην πραγματικότητα μέθοδος.

Εγκατάσταση της Ruby

Όπως θα έχετε παρατηρήσει, έχουμε μιλήσει για τη Ruby και τρέξει παραδείγματα, χωρίς καν να την εγκαταστήσουμε. Αυτό χάρη στην online έκδοση της irb. Στη σελίδα http://www.ruby-lang.org/en/downloads/ μπορείτε να κατεβάσετε τη

Ruby ή να την εγκαταστήσετε κατευθείαν από την κονσόλα στην περίπτωση που έχετε κάποιο unixοειδές λειτουργικό. Εάν θα δουλέψετε σε windows κατεβάστε την προτεινόμενη έκδοση. Η εγκατάσταση σε κάθε περίπτωση διαρκεί λίγα λεπτά, ανάλογα με το σύστημά σας και δεν παρουσιάζει ιδιαίτερη δυσκολία. Απλά πατάτε επόμενο.

Όπως θα δείτε, στο τέλος θα έχετε στο μηχάνημά σας τη μηχανή (engine) της Ruby μέσω της οποίας μπορείτε να τρέχετε τα προγράμματά σας, την irb που μπορείτε να την εκμεταλευτείτε για γρήγορους πειραματισμούς. Επίσης στα windows θα δείτε και τα «Start Command Prompt with Ruby» και «Ruby Gems Documentation Server». Το πρώτο είναι ένα κέλυφος (shell) μέσω της Ruby και προς το παρόν τουλάχιστον δε θα μας χρειαστεί. Το τελευταίο είναι ένας τοπικός server που μπορείτε να χρησιμοποιήσετε για να δείτε πληροφορίες για τα gems.

Πριν περάσουμε στα gems στην επόμενη ενότητα να πούμε ότι τώρα που έχετε τη Ruby στο σύστημά σας μπορείτε να τρέξετε τα αρχεία .rb και .rbw είτε με διπλό κλικ είτε ανοίγοντας την κονσόλα στο φάκελο που βρίσκεται το αρχείο (π.χ. hello_world.rb) και , ανεξαρτήτως του λειτουργικού συστήματος που έχετε, δώστε την εντολή «ruby hello_world.rb», χωρίς τα εισαγωγικά. Εάν το πρόγραμμά σας δέχεται και παραμέτρους τότε φυσικά με διπλό κλικ θα λάβετε μήνυμα λάθους, αφού δεν ορίζετε τις παραμέτρους. Έτσι καταφύγετε στην κονσόλα και γράψτε «ruby hello_world parameter1 parameter2» χωρίς τα εισαγωγικά και όπου parameter1 parameter2 τις κατάλληλες τιμές, προσέχοντας ταυτόχρονα τη σειρά και το πλήθος των παραμέτρων.

(Πολύ) Λίγα λόγια για τα Ruby Gems

Με τα Ruby gems προς το παρόν δε θα ασχοληθούμε. Θα σας πω μόνο ότι είναι οι βιβλιοθήκες της Ruby. Μπορείτε και σεις να φτιάξετε τη δική σας. Η κύρια σελίδα όπου μπορείτε να βρείτε gems είναι η http://rubygems.org/ . Ο τρόπος εγκατάστασης και η διαχείρησή τους θυμίζει linux. Π.χ. ανοίξτε το «Start Command Prompt with Ruby» που είδαμε προηγουμένως (σε unixοειδή λειτουργικά δε θα σας χρειαστεί) γράφοντας «gem install the_gem» εγκαθιστάται το gem με όνομα the_gem. Με την εντολή «gem list» μπορείτε να δείτε ποια gems έχετε αυτή τη στιγμή στο μηχάνημά σας. Μάλιστα μπορείτε να έχετε και πολλαπλές εκδόσεις του ίδιου gem.

Η ενσωμάτωση των gem στον κώδικά σας γίνεται με τις εξής 2 εντολές στην αρχή του αντίστοιχου αρχείου:

```
require 'rubygems'
require 'the_gem'
```

Η πρώτη εντολή λέει ότι θα χρειαστούμε gem και η δεύτερη ποιο ακριβώς. Υπόψη, ότι και το “ rubygems’” είναι ένα gem που αναλαμβάνει τη διαχείριση των gems!

Ακόμη, έχετε τη δυνατότητα να θέσετε κριτήρια για την έκδοση. Π.χ.

```
require 'rubygems'
gem 'activerecord', '>= 1.4.0'
gem 'actionpack', '<= 1.2.0'
gem 'actionmailer', '= 0.5.0'
gem 'rails', '= 0.9.3'
```

Έτσι, η Ruby θα ελένξει εάν το σύστημα στο οποίο τρέχει ο κώδικας έχει όχι μόνο τα συγκεκριμένα gems αλλά και κάποια έκδοση που να ικανοποιεί αυτά τα κριτήρια.

Έτσι, εάν για κάποιο λόγο χρησιμοποιείτε συγκεκριμένη έκδοση gem ενώ π.χ. η επόμενη δημιουργεί λάθη στο πρόγραμμα (λόγω π.χ. αλλαγών στο gem) ορίστε την έκδοση που θέλετε και ξενοιάστε.

Εάν σας ενδιαφέρει η Ruby ασχοληθείτε και με τα gems, άλλωστε θα το κάνετε αργά ή γρήγορα.

Loops στη Ruby

Στη Ruby εάν πηγαίνετε να γράψετε loop μην το κάνετε με τη λέξη «for»! Υπάρχει πιο κομψός τρόπος!

Καταρχήν στη Ruby θα βρείτε τη μέθοδο each. Π.χ. δοκιμάστε τον κώδικα (αρχείο ranges.rb)

```
puts "first instruction"
puts (1..5).class
puts "second instruction"
(1..5).each {|x| puts x}
puts "third instruction"
(1...5).each {|x| puts {x}}
puts "fourth instruction"
(1..5).each {|x| puts " #{x}"}
```

Στην έξοδο θα πάρετε:

```
first instruction
Range
second instruction
1
2
3
4
5
third instruction
 1
 2
 3
 4
fourth instruction
 1
 2
 3
 4
 5
```

Με αυτόν τον κώδικα σας μπλέκω με περισσότερα από ένα πράγματα οπότε ας τα πάρουμε με τη σειρά.

Καταρχήν, η εντολή puts εκτυπώνει στην κονσόλα πληροφορίες, είναι δηλαδή η αντίστοιχη της printf της C/C++ ή η System.out.println της Java κτλ. Προφανώς στη Ruby όπως και σε άλλες γλώσσες ότι είναι ανάμεσα σε διπλά ή απλά εισαγωγικά είναι String

(επίσης κλάση στη Ruby).

Δεύτερον, θα μιλήσουμε για την κλάση Range και τους τελεστές «..» και «...». Δείτε την εντολή «puts (1..5).class». Η κλάση Range όπως παρατηρείτε είναι μία κλάση που δηλώνει εύρος ακεραίων από μία ελάχιστη έως μία μέγιστη τιμή. Οι τελεστές 2 και 3 τελείες μπορούν να χρησιμοποιηθούν για τη δημιουργία αντικειμένων τύπου Range. Έτσι π.χ. το 1..5 σημαίνει από 1 έως 5 (δηλαδή 1<=x<=5) ενώ το 1...5 από 1 έως 4 (δηλαδή 1<=x<5).

Τρίτον, παρατηρείστε την εντολή (1..5).each {|x| puts x}. Χρησιμοποιούμε δηλαδή τις δύο κάθετες γραμμές || για να δηλώσουμε πως το κάθε στοιχείο θα το συμβολίζουμε με x και μετά ζητάμε την εκτύπωση «puts x». Σε μία δηλαδή γραμμή φτιάχνουμε το Range object και λέμε για κάθε ένα στοιχείο του εκτύπωσε την τιμή του.

Παρατηρήστε τώρα το αποτέλεσμα των εντολών «(1..5).each {|x| puts x}» και «(1...5).each {|x| puts x}».

Τέταρτον, ας δούμε την εντολή (1..5).each {|x| puts " #{x}"}. Εάν καταλάβατε τα παραπάνω τότε μπορούμε να επικεντρωθούμε στο " #{x}". Είπαμε ότι το # δηλώνει σχόλιο μίας γραμμής. Έχει όμως και έναν ακόμη ρόλο. Όπως παρατηρείτε γράφουμε μέσα σε διπλά εισαγωγικά το σύμβολο # και για την ακρίβεια γράφουμε "#{x}". Με αυτόν τον τρόπο κάνουμε γρήγορα μετατροπή ενός αριθμού, εδώ του x, σε String. Εάν και εδώ δε φαίνεται να έχει κάποια μεγάλη διαφορά, φανταστείτε τι θα πρέπει να γράψουμε εάν θέλαμε να γράψουμε στην κονσόλα με μία εντολή πολλές διαφορετικές μεταβλητές. Άλλες String, άλλες Fixnum κτλ. Θα μπορούσαμε να γράψουμε π.χ. puts x.to_s. Η to_s είναι η μέθοδος της Ruby αντίστοιχη της To_String που συναντάμε σε άλλες γλώσσες. Μπορούμε εναλλακτικά να χρησιμοποιήσουμε όμως τη μορφή " #{}" που είναι πιο συμπυκνωμένη.

Ας μην επεκταθώ άλλο. Μιλάμε για loops. Ένας λοιπόν τρόπος είναι η μέθοδος each που μπορούμε να τη χρησιμοποιήσουμε σε Ranges και Arrays. Υπάρχει όμως και ένας άλλος τρόπος. Δείτε τον κώδικα του αρχείου (for_in_ruby.rb):

```
#μετράμε από ένα εώς 5
1.upto(5) do |x|
        puts x
end
#μετράμε από 5 έως 1
5.downto(1) do |x|
        puts x
end
```

και την έξοδο:

```
1
2
3
4
5
5
```

```
4
3
2
1
```

Όπως είπαμε το 1 είναι αντικείμενο τύπου Fixnum, όπως και κάθε άλλος σταθερός αριθμός στη Ruby. Η κλάση αυτή έχει μεταξύ των άλλων και δύο μεθόδους την upto και την downto. Με αυτές μπορούμε να δημιουργούμε τους βρόχους επανάληψης for που συναντάμε σε άλλες γλώσσες. Παρατηρήστε ότι χρησιμοποιούμε τις λέξεις κλειδιά do και end. Εάν έχουμε πολλές εντολές μέσα στο βρόχο χρησιμοποιούμε αυτή τη δομή. Διαφορετικά μπορούμε να χρησιμοποιήσουμε και τις αγκύλες {} όπως κάναμε προηγουμένως με την εντολή each.

Η εντολή ελέγχου if

Στη Ruby θα συναντήσετε και την εντολή ελέγχου if. Η μορφή της είναι

```
if συνθήκη then
        ...
else
        ...
end
```

Δείτε τώρα τον κώδικα του αρχείου if.rb

```
a=3
b=4
if a==b then
        puts true
else
        puts false
end

puts true if a==b
```

Αρχικά ορίζουμε 2 μεταβλητές και εφόσον είναι ίσες εκτυπώνουμε true αλλιώς false. Σημειώστε ότι οι λέξεις true , false είναι επίσης αντικείμενα (δώστε την εντολή true.class ή false.class και βρείτε την κλάση ;-)) .

Όπως βλέπετε η if είναι η γνωστή μας if από άλλες γλώσσες. Στη Ruby όμως έχει μία ακόμα ιδιαιτερότητα. Βλέπετε ο Ασιάτης δημιουργός της Ruby παρατήρησε ότι πολύ συχνά σε προγράμματα θέλουμε να εκτελέσουμε μία μόνο εντολή εάν ισχύει μία συνθήκη. Έτσι έδωσε στην if και έναν δεύτερο τρόπο γραφής. Παρατηρήστε την τελευταία εντολή «puts true if a==b». Δηλαδή σε αυτή και μόνο σε αυτή την περίπτωση μπορούμε να γράψουμε πιο συμμαζεμένα τον κώδικά μας. Γράφουμε πρώτα το τι

θέλουμε να γίνει «puts true» και μετά τη συνθήκη «if a==b». Έτσι οι 3 γραμμές γίνονται μία γραμμή κώδικα.

Επίλογος

Κάπου εδώ θα σας αφήσω για να πειραματιστείτε. Ακόμα και αν δε θέλετε να εγκαταστήσετε τη Ruby δοκιμάστε την online irb για τα πειράματά σας. Στο επόμενο μάθημα θα μιλήσουμε για τις υπόλοιπες εντολές ελέγχου της Ruby καθώς επίσης και για το πως φτιάχνουμε συναρτήσεις.

Όπως βλέπετε η Ruby έχει κρατήσει τα καλύτερα χαρακτηριστικά από διάφορες γλώσσες προγραμματισμού μα το κυριότερο ότι στοχεύει στο να βοηθάει τον προγραμματιστή να γράφει όσον το δυνατό πιο συμμαζεμένο κώδικα. Έτσι στη Ruby είναι πιο εύκολο, σε σύγκριση με άλλες γλώσσες προγραμματισμού, να κατανοήσετε τον κώδικα τρίτων ανθρώπων.