# Learn to Program

By Chris Pine

Εγχειρίδιο για τη γλώσσα προγραμματισμού Ruby

**Μετάφραση στα ελληνικά:**
Ηλίας Μαργαριτίδης

Όταν προγραμματίζεις έναν υπολογιστή, πρέπει να "μιλάς" σε μια γλώσσα που ο υπολογιστής καταλαβαίνει: μια γλώσσα προγραμματισμού δηλαδή. Υπάρχουνε πάρα πολλές και διαφορετικών ειδών γλώσσες προγραμματισμού και αρκετές από αυτές είναι εξαιρετικές. Σε αυτό το εγχειρίδιο, διάλεξα να χρησιμοποιήσω την αγαπημένη μου γλώσσα προγραμματισμού, τη Ruby.

Πέρα από το γεγονός οτι είναι η αγαπημένη μου, η Ruby είναι επίσης η ευκολότερη γλώσσα προγραμματισμού που έχω δει (και έχω δει αρκετές). Στην πραγματικότητα αυτός είναι ο πραγματικός λόγος, για τον οποίο γράφω αυτό το εγχειρίδιο: Δεν αποφάσισα να γράψω ένα εγχειρίδιο και στη συνέχεια να διάλεξα την Ruby επειδή είναι η αγαπημένη μου. Αντί αυτού, βρήκα την Ruby, τόσο εύκολη, ώστε αποφάσισα οτι πραγματικά όφειλα ένα καλό εγχειρίδιο για αρχάριους. Είναι η απλότητα της Ruby που με παρακίνησε προς αυτό το εγχειρίδιο και όχι το γεγονός πως είναι η αγαπημένη μου γλώσσα. (Γράφοντας ένα παρόμοιο εγχειρίδιο χρησιμοποιώντας άλλη γλώσσα, όπως η C++, ή η Java, θα είχε απαιτήσει, εκατοντάδες εκατοντάδων σελίδες). Μην νομίσετε όμως πως η Ruby είναι μια γλώσσα αρχαρίων απλώς επειδή είναι εύκολη. Είναι μια δυνατή, επαγγελματικής αντοχής προγραμματιστική γλώσσα.

Όταν γράφουμε κάτι σε ανθρώπινη γλώσσα, αυτό που γράφεται καλείται κείμενο. Όταν γράφουμε κάτι σε μια γλώσσα υπολογιστή, αυτό που γράφεται καλείται κώδικας. Έχω περιλάβει πολλά παραδείγματα κώδικα Ruby μέσα στο εγχειρίδιο, τα περισσότερα από αυτά είναι ολοκληρωμένα προγράμματα, τα οποία μπορείτε να εκτελέσετε στον υπολογιστή σας. Για να κάνω τον κώδικα πιο ευανάγνωστο, έχω χρωματίσει διάφορα τμήματα του κώδικα με διαφορετικά χρώματα (για παράδειγμα, οι αριθμοί είναι πάντα πράσινοι). Οτιδήποτε υποτίθεται πως το πληκτρολογείτε θα είναι μέσα σε ένα άσπρο "κουτί", και οτιδήποτε "τυπώνει" ένα πρόγραμμα θα είναι σε ένα μπλε "κουτί".

Αν συναντήσετε κάτι, το οποίο δεν καταλαβαίνετε, ή αν έχετε οποιοδήποτε ερώτημα που δεν απαντήθηκε, καταγράψτε το και συνεχίστε το διάβασμα! Είναι αρκετά πιθανό οτι η απάντηση θα έρθει σε επόμενο κεφάλαιο. Παρόλα αυτά, αν η ερώτηση δεν απαντήθηκε από το τελευταίο κεφάλαιο, θα σας πω που μπορείτε να απευθύνετε την ερώτηση σας. Υπάρχουν αρκετοί υπέροχοι άνθρωποι εκεί έξω, κάτι περισσότερο από πρόθυμοι να βοηθήσουν. Το μόνο που χρειάζεστε είναι να ξέρετε που είναι.

Αλλά πρώτα χρειάζεται να κατεβάσετε και να εγκαταστήσετε την Ruby στον υπολογιστή σας.

## Εγκατάσταση σε Windows

Η εγκατάσταση της Ruby στα Windows είναι πολύ εύκολη. Πρώτα από όλα, χρειάζεται να κατεβάσετε τον "εγκαταστάτη" της Ruby. Υπάρχουν αρκετές εκδόσεις απότ ις οποίες μπορείτε να διαλέξετε. Αυτό το εγχειρίδιο χρησιμοποιεί την έκδοση 1.8.4, έτσι καλό είναι η έκδοση που θα κατεβάσετε να είναι τουλάχιστον μια μεταγενέστερη από την 1.8.4. (Προτιμότερο είναι να κατεβάσετε την πιο πρόσφατη έκδοση που είναι διαθέσιμη). Στη συνέχεια απλώς εκτελέστε το πρόγραμμα εγκατάστασης. Το πρόγραμμα θα σας ρωτήσει σε ποια τοποθεσία θέλετε να εγκαταστήσει την Ruby. Αν δεν έχετε κάποιον σημαντικό λόγο για κάτι διαφορετικό, απλά αφήστε την προεπιλεγμένη τοποθεσία εγκατάστασης.

Για να προγραμματίσετε, χρειάζεται να είστε ικανοί να γράψετε και να εκτελέσετε προγράμματα. Για να το κάνετε αυτό, θα χρειαστείτε έναν επεξεργαστή κειμένου και φυσικά το περιβάλλον γραμμής εντολών.

Ο εγκαταστάτης, περιλαμβάνει έναν υπέροχο επεξεργαστή κειμένου, με το όνομα SciTE

(The Scintilla Text Editor). Μπορείτε να εκτελέσετε τον SciTE, διαλέγοντας την αντίστοιχη επιλογή στο Μενού Έναρξης. Αν θα θέλατε ο κώδικας σας να χρωματίζεται, όπως στα παραδείγματα, του εγχειριδίου, τότε κατεβάστε τα παρακάτω αρχεία και αποθηκεύστε τα στον φάκελο εγκατάστασης του SciTE (c:/ruby/scite, αν επιλέξατε την προεπιλεγμένη τοποθεσία εγκατάστασης):

- Global Properties
- Ruby Properties

Επίσης θα ήταν καλή ιδέα να δημιουργήσετε έναν φάκελο κάπου, για να κρατήσετε όλα τα προγράμματα που θα δημιουργήσετε. Σιγουρέψτε, πως όταν αποθηκεύετε ένα πρόγραμμα, το αποθηκεύετε σε αυτόν τον φάκελο.

Για να μεταβείτε στην Γραμμή Εντολών, διαλέξτε την αντίστοιχη επιλογή, από τον "φάκελο" Βοηθήματα στο Μενού Έναρξη. Στη Γραμμή Εντολών θα θέλετε να πληγηθείτε στον φάκελο, όπου αποθηκεύετε τα προγράμματα σας. Πληκτρολογώντας: cd.. κάνετε τρέχων το ευρετήριο που βρίσκεται ένα επίπεδο πάνω (τον μητρικό δηλαδή) από το ευρετήριο εργασίας (τρέχων ευρετήριο), ενώ πληκτρολογώντας cd όνομα_ευρετηρίου, το ευρετήριο του οποίου πληκτρολογήσατε το όνομα, γίνεται το τρέχων ευρετήριο σας. Για να δείτε όλα τα ευρετήρια που βρίσκονται μέσα στο τρέχων ευρετήριο, πληκτρολογήστε dir/ad

Αυτό ήταν, τώρα έχετε ετοιμάσει ο,τι χρειάζεστε για να μάθετε να προγραμματίζετε.

**Εγκατάσταση σε Macintosh**

Αν έχετε λειτουργικό σύστημα Mac OS X 10.2 (Jaguar), τότε έχετε και την Ruby, εγκατεστημένη στον υπολογιστή σας. Τι θα μπορούσε να είναι ευκολότερο; Δυστυχώς δεν είναι πιθανό οτι θα μπορέσετε να χρησιμοποιήσετε την Ruby, στο Mac OS X 10.1 και οποιαδήποτε παλιότερη έκδοση.

Για να προγραμματίσετε, χρειάζεται να είστε σε θέση να γράψετε και να εκτελέσετε προγράμματα. Για να το υλοποιήσετε αυτό, θα χρειαστείτε έναν επεξεργαστή κειμένου και μια γραμμή εντολών.

Η γραμμή εντολών είναι προσβάσιμη, μέσω της Εφαρμογής Τερματικού - Terminal application- (βρίσκεται στις Εφαρμογές/Εργαλεία).

Ως επεξεργαστή κειμένου, μπορείτε να χρησιμοποιήσετε, οποιονδήποτε, με τον οποίο είστε εξοικειωμένοι και νιώθετε άνετα στη χρήση του. Ωστόσο αν χρησιμοποιήσετε τον TextEdit, φροντίστε να αποηκεύετε τα αρχεία σας με την μορφή: text-only. Σε διαφορετική περίπτωση τα προγράμματα σας δεν θα λειτουργήσουν. Άλλες επιλογές για να προγραμματίσετε είναι οι επεξεργαστές, emacs, vi και pico, οι οποίοι είναι όλοι προσβάσιμοι από την γραμμή εντολών.

Αυτό ήταν, τώρα έχετε ετοιμάσει ο,τι χρειάζεστε για να μάθετε να προγραμματίζετε.

**Εγκατάσταση σε Linux**

Η πρώτη δουλειά σας είναι να ελέγξετε αν η Ruby, είναι ήδη εγκατεστημένη. Πληκτρολογήστε: which ruby. Αν το μήνυμα είναι κάτι σαν το παρακάτω: /usr/bin/which: no ruby in (...), τότε χρειάζεται να "κατεβάσετε" την Ruby. Σε διαφορετική περίπτωση (πχ ένα μήνυμα: /usr/bin/ruby), ελέγξτε την έκδοση της Ruby, που είναι εγκατεστημένη με την εντολή: ruby -v. Αν η έκδοση είναι παλιότερη, από την τελευταία σταθερή διαθέσιμη στην ιστοσελίδα που δόθηκε παραπάνω, τότε ίσως να θέλετε να την αναβαθμίσετε.

Αν είστε ο root user, τότε πιθανά, δεν χρειάζεστε καθόλου οδηγίες για να εγκαταστήσετε την

Ruby. Αν όμως δεν είστε ο root user, , τότε πιθανά χρειαστεί να ζητήσετε από τον διαχειριστή του συστήματος, να την εγκαταστήσει για εσάς.(Έτσι όλοι οι χρήστες του συστήματος θα μπορούν να χρησιμοποιήσουν την Ruby). Διαφορετικά, μπορείτε να την εγκαταστήσετε κι εσείς, αλλά μόνο εσείς θα μπορείτε να την χρησιμοποιήσετε . Μετακινήστε το αρχείο που κατεβάσατε σε ένα προσωρινό ευρετήριο (πχ $HOME/tmp). Αν το όνομα του αρχείου είναι ruby-1.6.7.tar.gz μπορείτε να το ανοίξετε δίνοντας την εντολή: tar zxvf ruby-1.6.7.tar.gz. Με την εντολή αυτή δημιουργείται ένα ευρετήριο με το όνομ ruby-1.6.7. Κάντε τρέχων το συγκεκριμένο ευρετήριο, που μόλις δημιουργήσατε (στο παράδειγμα μας, με την εντολή cd ruby-1.6.7).

"Ρυθμίστε" την εγκατάσταση σας πληκτρολογώντας: ./ configure --prefix=$HOME. Στη συνέχεια πληκτρολογήστε: make. Με την εντολή αυτή "χτίζετε" τον διερμηνευτή της Ruby. Αυτή η εργασία ίσως διαρκέσει μερικά λεπτά. Μετά την ολοκλήρωση της, πληκτρολογήστε: make install για να τον εγκαταστήσετε.

Στη συνέχεια προσθέστε τη διαδρομή $HOME/bin στο μονοπάτι εντοπισμού εντολών. Για να γίνει αυτό πρέπει να επεξεργαστείτε το αρχείο:$HOME/.bashrc και να κάνετε την σχετική προσθήκη. (Για την ενεργοποίηση της αλλαγής πιθανά να χρεαστεί να κάνετε επανεκκίνηση). Στη συνέχεια, μπορείτε να ελέγξετε την εγκατάσταση σας με την εντολή: ruby -v. Αν η απάντηση του συστήματος είναι η έκδοση της Ruby που μόλις εγκαταστήσατε (πχ ruby 1.8.7 (2008-08-11 patchlevel 72) [i586-linux]), μπορείτε πλέον να διαγράψετε το αρχείο της εγκατάστασης.

Σημείωση του μεταφραστή: Πολλές διανομές Linux σας δίνουν την δυνατότητα να εγκαταστήσετε εύκολα την Ruby και πολλές από τις βιβλιοθήκες της και τις προεκτάσεις της, μέσα από τους διαχειριστές πακέτων που έχουν (πχ στα OpenSuse μέσα από την διαχειριστή Yast2) .

Αυτό ήταν, τώρα έχετε ετοιμάσει ο,τι χρειάζεστε για να μάθετε να προγραμματίζετε.

**Κεφάλαιο 1: Αριθμοί**

Αφού έχετε ολοκληρώσει όλε τις ρυθμίσεις, ας γράψουμε επιτέλους ένα πρόγραμμα. Ανοίξτε τον αγαπημένο σας επεξεργαστή κειμένου και πληκτρολογήστε:

```
puts 1+2
```

Αποθηκεύστε το πρόγραμμα σας (ναι, αυτό είναι ένα πρόγραμμα) με το όνομα calc.rb (η επέκταση **.rb** είναι η πιο συνηθισμένη για τα προγράμματα που είναι γραμμένα σε Ruby). Τώρα εκτελέστε το πρόγραμμα σας, πληκτρολογώντας ruby calc.rb στην γραμμή εντολών. Θα πρέπει να εμφανιστεί ένα 3 στην οθόνη σας. Είδατε; το να προγραμματίζετε δεν είναι τόσο δύσκολο.

**Εισαγωγή στη μέθοδο puts**

Τι συμβαίνει λοιπόν σε αυτό το πρόγραμμα; Είμαι σιγουρός πως μπορείτε να βρείτε πόσο κάνει 1+2. Το πρόγραμμα μας βασικά έχει το ίδιο αποτέλεσμα με το επόμενο:

```
puts 3
```

Η εντολή puts, απλώς εμφανίζει στην οθόνη, οτιδήποτε την ακολουθεί

**Ακέραιοι και Δεκαδικοί**

Στις περισσότερες γλώσσες προγραμματισμού (και η Ruby δεν αποτελεί εξαίρεση) οι αριθμοί χωρίς δεκαδικά ψηφία ονομάζονται ακέραιοι και οι αριθμοί με δεκαδικά ψηφία, ονομάζονται αριθμοί κινητής υποδιαστολής ή πιο απλά δεκαδικοί ή πραγματικοί.

Ακολουθούν μερικοί ακέραιοι:

```
5
-205
99999999999999999999999999
0
```

Και στη συνέχεια ορισμένοι πραγματικοί:

```
54.321
0.001
-205.3884
0.0
```

Στην πράξη, τα περισσότερα προγράμματα δεν χρησιμοποιούν πραγματικούς, αλλά μόνο ακέραιους (εξάλλου ποιος ενδιαφέρεται για 7.4 e-mails ή πλοηγείται σε 1.8 ιστοσελίδες ή ακούει 5.24 τραγούδια. Οι πραγματικοί αριθμοί χρησιμοποιούνται περισσότερο για ακαδημαϊκούς σκοπούς (πειράματα φυσικής και άλλα παρόμοια) και για τρισδιάστατα γραφικά . Ακόμη και τα περισσότερα προγράμματα που χρησιμοποιούν δεδομένα για χρηματικά ποσά, χρησιμοποιούν ακέραιους και απλώς παρακολουθούν την ποσότητα των λεπτών.

Απλή αριθμητική

Μέχρι τώρα έχουμε δει όλα όσα μπορούν να φτιάξουν έναν απλό αριθμητικό υπολογιστή. (Οι αριθμητικοί υπολογιστές, πάντα χρησιμοποιούν δεκαδικούς αριθμούς, έτσι αν θέλετε ο υπολογιστής σας να λειτουργεί ως αριθμητικός, θα πρέπει να χρησιμοποιήσετε δεκαδικούς). Ως σύμβολα της πρόσθεσης και της αφαίρεσης χρησιμοποιείται το + και το – αντίστοιχα, ενώ για τον πολλαπλασιασμό και τη διαίρεση χρησιμοποιούνται το * και το /. Τα περισσότερα πληκτρολόγια έχουνε αυτά τα πλήκτρα στο αριθμητικό τμήμα στην δεξιά πλευρά. Αν έχετε μικρότερο πληκτρολόγιο ή φορητό υπολογιστή, χρησιμοποιείτε τον συνδυασμό πλήκτρων shift και 8 για τον πολλαπλασιασμό ενώ για την διαίρεση είναι η κάθετος (/) που βρίσκεται μαζί με το αγγλικό ερωτηματικό(?). Ας προσπαθήσουμε να επεκτείνουμε λιγάκι το πρόγραμμα calc.rb. Πληκτρολογήστε τις παρακάτω γραμμές και εκτελέστε το μετά:

```
puts 1.0 + 2.0
puts 2.0 * 3.0
puts 5.0 - 8.0
puts 9.0 / 2.0
```

Ας δούμε τι εμφανίζεται στην οθόνη μας:

```
3.0
6.0
-3.0
```

## 4.5

Τα κενά διαστήματα μέσα στο πρόγραμμα δεν είναι σημαντικά. Απλώς κάνουν τον κώδικα πιο ευανάγνωστο. Ας δούμε τις ίδιες πράξεις με ακέραιους:

```
puts 1+2
uts 2*3
puts 5-8
puts 9/2
```

Τα περισσότερα αποτελέσματα, είναι ίδια σωστά?

```
3
6
-3
4
```

Χμ, εξαιρείται το τελευταίο και ο λόγο είναι πως όταν κάνετε πράξεις με ακέραιους το αποτέλεσμα που θα πάρετε θα είναι πάλι ακέραιος. Όταν ο υπολογιστής δεν μπορεί να επιστρέψει το σωστό αποτέλεσμα, τότε πάντα το στρογγυλοποιεί στον πλησιέστερο προς τα κάτω ακέραιο. (Φυσικά το τέσσερα είναι η σωστή ακέραιη απάντηση για τη διαίρεση 9/2. Ίσως βέβαια να μην ήταν η απάντηση που προσδοκούσατε

Ίσως κάποιοι από εσάς, αναρωτιούνται σε τι χρησιμεύει η ακέραιοη διαίρεση. Ας πούμε οτι θέλετε να πάτε στον κινηματογράφο και έχετε 36 ευρώ στηντ σέπη σας. Το εισητήριο για μαι ταινία κοστίζει 8 ευρώ. Πόσες ταινίες μπορείτε να παρακολουθήσετε; 9/2... 4 ταινίες. Η απάντηση 4,5 σίγουρα δεν είναι σωστή σε αυτή την περίπτωση. Δεν υπάρχει περίπτωση να σας αφήσουν να παρακολουθήσετε την μισή ταινία, ή να επιτρέψουν να εισέλθει στην αίθουσα ο μισός εαυτός σας. Ορισμένα πράγματα, απλά δεν είναι διαιρέσιμα.

Τώρα καλό είναι να πειραματιστείτε μόνοι σας με μερικά προγράμματα. Αν θέλετε να χρησιμοποιήσετε πιο πολύπλοκες εκφράσεις, μπορείτε να χρησιμοποιήσετε παρενέσεις. Για παράδειγμα:

```
puts 5 * (12-8) + -15
puts 98 + (59872 / (13*8)) * -52
```

```
5
-29802
```

**Προσπαθήστε ορισμένα πράγματα:**

Γράψτε ένα πρόγραμμα που σας λέει:

- Πόσες ώρες υπάρχουν σε ένα έτος?
- Πόσα λεπτά υπάρχουν σε μια δεκαετία;
- Πόσων δευτερολέπτων είστε; (δηλαδη η ηλικία σας σε δευτερόλεπτα)
- Πόσες σοκολάτες ελπίζετε να φάτε στη ζωή σας;

  **Προειδοποίηση:** *Αυτό το πρόγραμμα θα χρειαστεί λίγο χρόνο περισσότερο για να κάνει τον υπολογισμό*

Και τέλος έα κάπως δυσκολότερο ερώτημα:

- Αν είμαι 1031 εκατομμυρίων δευτερολέπτων, τότε πόσο χρονών είμαι;

Όταν τελειώσετε τα παιχνιδίσματα με τους αριθμούς, τότε προχωρήστε στα "γράμματα" (χαρακτήρες).

## 2. Γράμματα (χαρακτήρες).

Μάθαμε πλέον ο,τι σχετίζεται με τους αριθμούς, αλλά τι γίνεται με τους χαρακτήρες; τις λέξεις; τα κείμενα; Αναφερόμαστε σε ομάδες γραμμάτων, μέσα σε ένα πρόγραμμα ως αλφαριθμητικά (μπορείτε να τα σκέφτεστε ως τυπωμένα γράμματα τα οποία είναι ομαδοποιημένα σε ένα πλαίσιο). Για να διευκολυνθείτε να διακρίνετε ποιο τμήμα του κώδικα ένα αλφαριθμητικό, θα χρωματίζω τα αλφαριθμητικά, κόκκινα. Παρατίθενται ορισμένα αλφαριθμητικά:

```
'Hello.'
'Ruby rocks.'
'5 is my favorite number... what is yours?'
'Snoopy says #%^?&*@! when he stubs his toe.'
'    '
''
```

Όπως μπορείτε να δείτε, τα αλφαριθμητικά, μπορούν να έχουν σημεία στίξης, αριθμητικούς χαρακτήρες, σύμβολα, και κενά μεταξύ των χαρακτήρων, δηλαδή κάτι παραπάνω από απλά γράμματα. Το τελευταίο αλφαριθμητικό, δεν έχει τίποτα. Μια τέτοια περίπτωση αλφαριθμητικού, θα μπορούσε να χαρακτηριστεί άδειο αλφαριθμητικό.

Μέχρι τώρα έχουμε χρησιμοποιήσει την εντολή puts για να εμφανίσουμε αριθμούς, ας την δοκιμάσουμε και με μερικά αλφαριθμητικά.

```
puts 'Hello, world!'
puts ''
puts 'Good-bye.'
```

```
Hello, world!

Good-bye.
```

Μια χαρά δούλεψε και με τα αλφαριθμητικά. Δοκιμάστε τώρα μερικά δικά σας αλφαριθμητικά για εξάσκηση.

### Αριθμητικές πράξεις αλφαριθμητικών

Όπως ακριβώς μπορείτε να κάνετε αριθμητικές πράξεις με τα νούμερα, μπορείτε να κάνετε αριθμητικές πράξεις και με τα αλφαριθμητικά. Δηλαδή ένα είδος πράξεων. Πάντως μπορείτε να προσθέσετε δύο αλφαριθμητικά. Ας προσπαθήσουμε να προσθέσουμε δύο αλφαριθμητικά και να δούμε τι κάνει η εντολή puts με αυτά.

```
puts 'I like' + 'apple pie.'
```

```
I likeapple pie.
```

Βλέπετε οτι λείπει ένας κενός χαρακτήρας ανάμεσα στο 'I like' και το 'apple pie.' Τα κενά συνήθως δεν έχεις καμία σημασία, έχουνε σημασία όμως μέσα στα αλφαριθμητικά. Είναι γνωστό πως οι υπολογιστές δεν κάνουνε αυτά που θέλετε να κάνουνε, αλλά αυτά που τους λέτε να κάνουνε. Ας βελτιώσουμε το προηγούμενο παράδειγμα:

```
puts 'I like ' + 'apple pie.'
puts 'I like' + ' apple pie.'
```

```
I like apple pie.
I like apple pie.
```

Όπως βλέπετε δεν έχει σημασία σε ποιο από τα δύο αλφαριθμητικά προσθέσατε τον κενό χαρακτήρα.

Εκτός από την πρόσθεση μπορείτε να πραγματοποιήσετε και πολλαπλασιασμούς με τα αλφαριθμητικά. Πολλαπλασιασμό με έναν αριθμό όμως. Δείτε αυτό:

```
puts 'blink ' * 4
```

```
blink blink blink blink
```

Αν σας παραξενεύει, αυτό δεν θα έπρεπε, είναι απόλυτα λογικό. Εξάλλου, 7*3 πραγματικά σημαίνει απλά 7+7+7, έτσι η πράξη: 'moo'*3 απλά σημαίνει 'moo'+'moo'+'moo'.

**12 εναντίον '12'**

Πριν προχωρήσουμε περισσότερο, θα πρέπει να βεβαιωθούμε πως έχει γίνει κατανοητή η διαφορά ανάμεσα στους αριθμούς και τους χαρακτήρες. Το 12 είναι ένας αριθμός, αλλά το '12' είναι ένα αλφαριθμητικό με δύο χαρακτήρες, δύο ψηφία.

Ας πειραματιστούμε με αυτές τις διαφορές για λίγο:

```
puts  12  +  12
puts '12' + '12'
puts '12  +  12'
```

```
24
1212
12  +  12
```

Για δείτε κι εδώ τις διαφορές

```
puts  2  *  5
puts '2' *  5
puts '2  *  5'
```

```
10
22222
2  *  5
```

Τα παραδείγματα ήταν απλά και ξεκάθαρα. Να θυμάστε πως αν δεν είστε προσεκτικοί στην ανάμιξη αριθμών και αλφαριθμητικών, είναι πολύ πιθανό να αντιμετωπίσετε προβλήματα.

**Προβλήματα**

Αν μέχρι αυτό το σημείο, δεν έχετε δοκιμάσει κάποια πράγματα που δεν δούλεψαν, είναι μια καλή ευκαιρία να το κάνετε. Ορίστε λοιπόν ορισμένες προβληματικές καταστάσεις:

```
puts '12' + 12
puts '2' * '5'
```

```
#<TypeError: can't convert Fixnum into String>
```

Να επιτέλους ένα μήνυμα λάθους. Τα προβλήματα εδώ είναι πως δεν μπορείτε να προσθέσετε ένα αλφαριθμητικό με έναν αριθμό ή να πολλαπλασιάσετε δύο αλφαριθμητικά μεταξύ τους.

Επίσης δεν έχει κανένα απολύτως νόημα να γράψετε κάτι τέτοιο: I

```
puts 'Betty' + 12
puts 'Fred' * 'John'
```

Κάτι ακόμη που είναι χρήσιμο να γνωρίζετε: Μπορείτε να γράψετε 'pig'*5 σε ένα πρόγραμμα, αφού αυτό απλά σημαίνει 5 φορές το αλφαριθμητικό 'pig' η μία μετά την άλλη. Όμως δεν μπορείτε να γράψετε:5*'pig', γιατί αυτό σημαίνει 'pig' φορές το 5, κάτι το οποίο δεν έχει κανένα νόημα.

Ένα τελευταίο θέμα. Τι θα πρέπει να γράψω για να τυπωθεί το μήνυμα: You 're swell!

Ας δοκιμάσουμε τον παρακάτω κωδίκα που σκεστόσασταν λογικά:

```
puts 'You're swell!'
```

Όμως αυτό δεν θα δουλέψει. Μην προσπαθήσετε καν να εκτελέσετε το πρόγραμμα. Ο υπολογιστής θα νομίσει οτι τελειώσατε με το αλφαριθμητικό στο  You. (να ένας ακόμη λόγος που είναι χρήσιμο να χρησιμοποιείτε έναν επεξεργαστή κειμένου που χρωματίζει τον κώδικα με συντακτικά κριτήρια). Έτσι πως τελικά θα επιτρέψετε στον υπολογιστή να καταλάβει οτι θέλετε να παραμείνετε μέσα στο αλφαριθμητικό; Μπορούμε να κάνουμε τον υπολογιστεί να αγνοήσει την απόστροφο με τον εξής τρόπο:

```
puts 'You\'re swell!'
```

```
You're swell!
```

Η κάθετος (backslash) θεωρείται χαρακτήρας διαφυγής. Όχι όμως για όλους τους χαρακτήρες. Με άλλα λόγια, αν έχετε μια κάθετο και μετά έναν άλλο χαρακτήρα, θα εμφανιστούν κανονικά και η κάθετος και ο,τι ακολουθεί. Αν όμως ακολουθεί απόστροφος ή κάθετος (δηλαδή να υπάρψουν δύο κάθετοι στη σειρά), τότε η κάθετος λειτουργεί ως χαρακτήρας διαφυγής.

Ακολουθούν μερικά παραδείγματα για να γίνει κατανοητό:

```
puts 'You\'re swell!'
puts 'backslash at the end of a string:  \\'
```

```
puts 'up\\down'
puts 'up\down'
```

```
You're swell!
backslash at the end of a string:  \
up\down
up\down
```

Από τη στιγμή που η κάθετος δεν λειτουργεί ως χαρακτήρας διαφυγής με το 'd', αλλά λειτουργεί με τον ίδιο του τον εαυτό, τότε τα δύο τελευταία παραδείγματα αλφαριθμητικών είναι ακριβώς τα ίδια. Δεν μοιάζουν ίδια στον υπολογιστή, αλλά στην εκτέλεση είναι πραγματικά ίδια .

Αν μέχρι τώρα έχετε απορίες, απλά συνεχίστε να διαβάζετε.

## 3. Μεταβλητές και εκχώρηση

Μέχρι τώρα, όποτε χρησιμοποιήθηκε η puts με αλφαριθμητικά ή αριθμούς, οι τιμές που χρησιμοποιήθηκαν, χάνονταν. Αυτό σημαίνει πως χρειαζόταν να τυπωθεί κάτι δυο φορές, θα έπρεπε να πληκτρολογηθεί επίσης δυο φορές

```
puts '...you can say that again...'
puts '...you can say that again...'
```

```
...you can say that again...
...you can say that again...
```

Θα ήταν πολύ καλύτερο αν ήταν εφικτό να πληκτρολογηθεί μια φορά και στη συνέχεια να διατηρηθεί να αποθηκευθεί κάπου. Ε λοιπόν είναι εφικτό.

Για να αποθηκεύσετε το αλφαριθμητικό στην μνήμη του υπολογιστή σας, χρειάζεται να δώσετε ένα όνομα στο αλφαριθμητικό. Οι προγραμματιστές, συχνά αναφέρονται σε αυτή την διαδικασία ως εκχώρηση, και τα ονόματα που δίνουνε στα αλφαριθμητικά τα καλούν, μεταβλητές. Οι μεταβλητές μπορούν να είναι μια οποιαδήποτε σειρά γραμμάτων και αριθμών, αλλά ο πρώτος χαρακτήρας πρέπει να είναι οπωσδήποτε γράμμα και μάλιστα πεζό (τα σύμβολα δεν επιτρέπονται)

Ας δοκιμάσουμε ξανά το τελευταίο πρόγραμμα, αλλά αυτή τη φορά θα δώσουμε στο αλφαριθμητικό το όνομα myString (Θα μπορούσαμε να το είχαμε ονομάσει επίσης str ή myOwnLittleString ή henryTheEighth).

```
myString = '...you can say that again...'
puts myString
puts myString
```

```
...you can say that again...
...you can say that again...
```

Όποτε προσπαθήσατε να κάνετε μία ενέργεια στη μεταβλητή myString, αυτή εφαρμόστηκε στο αλφαριθμητικό '...you can say that again...'. Φανταστείτε το σαν η μεταβλητή myString να

δείχνει προς το αλφαριθμητικό ...you can say that again...'.

Ορίστε ένα κάπως πιο ενδιαφέρον παράδειγμα:

```
name = 'Patricia Rosanna Jessica Mildred Oppenheimer'
puts 'My name is ' + name + '.'
puts 'Wow!  ' + name + ' is a really long name!'
```

```
My name is Patricia Rosanna Jessica Mildred Oppenheimer.
Wow!  Patricia Rosanna Jessica Mildred Oppenheimer is a really long name!
```

Όπως ακριβώς εκχωρείτε ένα αντικείμενο σε μια μεταβλητή, μπορείτε να ξαναεκχωρήσετε ένα άλλο αντικείμενο στην ίδια μεταβλητή. Γι' αυτό ακριβώς τις ονομάζουμε και μεταβλητές. Επειδή η τιμή η οποία αποθηκεύτηκε σε αυτές, μπορεί να αλλάξει.

```
composer = 'Mozart'
puts composer + ' was "da bomb", in his day.'
composer = 'Beethoven'
puts 'But I prefer ' + composer + ', personally.'
```

```
Mozart was "da bomb", in his day.
But I prefer Beethoven, personally.
```

Φυσικά οι μεταβλητές μπορούν να δείχνουν σε οποιασδήποτε κατηγορίας αντικείμενα (δεδομένα) και όχι μόνο αλφαριθμητικά

```
var = 'just another ' + 'string'
puts var

var = 5 * (1+2)
puts var
```

```
just another string
15
```

Στην πραγματικότητα οι μεταβλητές μπορούν να δείχνουν σε οτιδήποτε, ακόμη και άλλες μεταβλητές.  Δοκιμάστε και δείτε τι γίνεται, όταν συμβαίνει αυτό:

```
var1 = 8
var2 = var1
puts var1
puts var2

puts ''

var1 = 'eight'
puts var1
puts var2
```

```
8
```

```
8

eight
8
```

Έτσι αρχικά αφού προσπαθήσαμε η μεταβλητή var2 να δείχνει προς την var1, αυτή αντί αυτού έδειξε στο νούμερο 8, δηλαδή στην τιμή που έδειχνε (ή που είχε αποθηκευμένη) και η var1. Στη συνέχεια, είχαμε την var1 να δείχνει προς το 'eight'. Στη συνέχεια βάλαμε την var1 να δείχνει στο 'eight', αλλά από την στιγμή που η var2 δεν έδειξε ποτέ προς την var1, παραμένει να δείχνει στο 8.

Αφού μάθατε τις μεταβλητές, τους αριθμούς και τα αλφαριθμητικά καιρός να μάθετε να τα αναμιγνύετε και να τα συνδυάζετε.

**4. Αναμιγνύοντας τα**

Μέχρι τώρα είδαμε μερικά διαφορετικά είδη αντικειμένων (αριθμούς και χαρακτήρες), δημιουργήσαμε μεταβλητές να δείχνουν προς αυτά τα αντικείμενα και το επόμενο πράγμα πυο θέλουμε να κάνουμε είναι να τα βάλουμε να συνεργάζονται όλα μαζί.

Από αυτά που έχουν ειπωθεί μέχρι τώρα, είναι φανερό πως αν θέλουμε ένα πρόγραμμα να τυπώσει το νούμερο 25 τότε ο τρόπος που ακολουθεί δεν είναι ο κατάλληλος, γιατί δεν είναι δυνατό να προστεθούν αριθμοί και αλφαριθμητικά:

```
var1 = 2
var2 = '5'
puts var1 + var2
```

Μέρος του προβλήματος είναι οτι ο υπολογιστής σας δεν μπορεί να καταλάβει αν προσπαθείτε να πάρετε σαν αποτέλεσμα το 7 (2 + 5) ή το 25 ('2' + '5')

Πριν να είναι εφικτό να προσθέσουμε τα δύο προηγούμενα αντικείμενα, χρειάζεται με κάποιο τρόπο να λάβουμε την αλφαριθμητική εκδοχή της var1 ή την αριθμητική εκδοχή της var2

**Μετατροπές**

Για να πάρουμε την αλφαριθμητική "εκδοχή" ενός αντικειμένου, απλώς γράφουμε .to_s μετά από αυτό:

```
var1 = 2
var2 = '5'
puts var1.to_s + var2
```

```
25
```

Ομοίως γράφοντας: .to_i παίρνουμε την ακέραιη εκδοχή ενός αντικειμένου, ενώ γράφοντας: .to_f παίρνουμε την πραγματική (δεκαδική) εκδοχή. Ας κοιτάξουμε λίγο πιο προσεκτικά τι κάνουνε (και τι δεν κάνουνε) αυτές οι τρεις μέθοδοι:

```
var1 = 2
var2 = '5'
puts var1.to_s + var2
puts var1 + var2.to_i
```

```
25
7
```

Σημειώστε πως ακόμα και αφού έχουμε πάρει την αλφαριθμητική εκδοχή της μεταβλητής var1 με την προσθήκη του .to_s, η μεταβλητή var1 αυτή καθεαυτή συνεχίζει να δείχνει στο 2 και ποτέ στο '2'

Αν δεν εκχωρήσουμε και πάλι ρητά κάποια τιμή στην var1 (θυμίζουμε πως για την εκχώρηση απαιτείται ένα =), τότε αυτή θα δείχνει στο 2 για όλη τη διάρκεια του προγράμματος.

Τώρα προσπαθήστε μερικές ακόμη (και κάποιες από αυτές κάπως παράξενες) μετατροπές.

```
puts '15'.to_f
puts '99.999'.to_f
puts '99.999'.to_i
puts ''
puts '5 is my favorite number!'.to_i
puts 'Who asked you about 5 or whatever?'.to_i
puts 'Your momma did.'.to_f
puts ''
puts 'stringy'.to_s
puts 3.to_i
```

```
15.0
99.999
99

5
0
0.0

stringy
3
```

Τα προηγούμενα αποτελέσματα ίσως σας δημιούργησαν μερικές απορίες. Το πρώτο αποτέλεσμα είναι κάτι φυσιολογικό σύμφωνα με όσα είπαμε. Στη συνέχεια, μετατρέπεται το αλφαριθμητικό '99.999' σε έναν πραγματικό αριθμό και μετά σε έναν ακέραιο. Ο πραγματικός ήταν αυτό που περιμέναμε, ενώ ο ακέραιος όπως πάντα ήταν στρογγυλοποιημένος προς τα κάτω.

Στη συνέχεια ακολούθησαν μερικά παραδείγματα, μη συνηθισμένων αλφαριθμητικών τα οποία μετατράπηκαν σε αριθμούς. Η προσθήκη .to_i αγνοεί το αλφαριθμητικό μετά το πρώτο

πράγμα το οποίο δεν κατανοεί. Έτσι στο πρώτο αλφαριθμητικό, μετατράπηκε το 5, αλλά το υπόλοιπο αλφαριθμητικό από τη στιγμή που ξεκινήσανε τα γράμματα αγνοήθηκε . Τα επόμενα δύο αλφαριθμητικά αγνοήθηκαν ολόκληρα από τη στιγμή που ξεκινούσαν με γράμματα. Έτσι ο υπολογιστής τους δίνει την τιμή μηδέν.

Τέλος, οι δύο τελευταίες μετατροπές δεν κάνανε τίποτα απολύτως, όπως ήταν αναμενόμενο.

**Μια δεύτερη ματιά στην εντολή puts**

Υπάρχει κάτι παράξενο στην μέχρι τώρα αγαπημένη μας μέθοδο. Ρίξτε μια ματιά στα παρακάτω:

```
puts 20
puts 20.to_s
puts '20'
```

```
20
20
20
```

Γιατί και στις τρεις παραπάνω περιπτώσεις, το αποτέλεσμα που τυπώνεται είναι το ίδιο; Οι δύο τελευταίες έτσι θα έπρεπε αφού το 20.to_s ισοδυναμεί με το '20'. Τι συμβαίνει όμως στην πρώτη περίπτωση με τον ακέραιο αριθμό 20; Τι ακριβώς σημαίνει να γράψει κάποιος τον ακέραιο αριθμό 20; Όταν γράφετε σε ένα χαρτί ένα 2 και ένα 0 γράφετε ένα αλφαριθμητικό και όχι έναν ακέραιο. Ο ακέραιος 20 μετράει συγκεκριμένες ποσότητες όπως τα δάχτυλα του ανθρώπου και δεν είναι απλώς ένα 2 που ακολουθείται από ένα 0.

Εδώ λοιπόν βρίσκεται και το μεγάλο μυστικό της μεθόδου, puts: Πριν να προσπαθήσει η puts να γράψει ένα αντικείμενο, χρησιμοποιεί το .to_s για να πάρει την αλφαριθμητική εκδοχή αυτού του αντικειμένου. Για την ακρίβεια η κατάληξη s στην λέξη puts σημαίνει string , δηλαδή αλφαριθμητικό. Δηλαδή στην πραγματικότητα puts σημαίνει put string.

Ίσως αυτό να μην σας φαίνεται πολύ συναρπαστικό τώρα, αλλά υπάρχουν πολλών πολλών ειδών αντικείμενα στην Ruby (μάλιστα μπορείτε να κάνετε και μόνοι σας και θα μάθετε πως) και είναι πολύ ωραίο να ξέρετε τι θα συμβεί, αν προσπαθήσετε να χρησιμοποιήσετε τη μέθοδο puts με ένα πολύ παράξενο αντικείμενο όπως μια εικόνα της γιαγιάς σας ή ένα αρχείο μουσικής ή κάτι άλλο. Αυτά όμως θα τα μάθετε αργότερα.

Στο μεταξύ, είστε σε θέση να μάθετε μερικές ακόμα μεθόδους, οι οποίες θα σας επιτρέψουν να γράψετε πολλών ειδών διασκεδαστικά προγράμματα.

**Οι μέθοδοι gets και chomp**

Αν puts σημαίνει put string, σίγουρα μπορείτε να υποθέσετε τι σημαίνει το gets, και όπως ακριβώς η puts εμφανίζει ένα αλφαριθμητικό, η gets δέχεται ως είσοδο (ανακτά) ένα αλφαριθμητικό .Που τα βρίσκει όμως και τα παίρνει;

Από εσάς, ή για την ακρίβεια από το πληκτρολόγιο σας. Αφού μόνο το πληκτρολόγιο σας

δημιουργεί αλφαριθμητικά, τα οποία δουλεύουν καλά. Αυτό που συμβαίνει στην πραγματικότητα είναι οτι η gets αναμένει και διαβάζει οτι πληκτρολογείτε, μέχρι να πατήσετε το πλήκτρο Enter. Δοκιμάστε το:

```
puts gets
```

```

Is there an echo in here?
Is there an echo in here?
```

Φυσικά, οτιδήποτε πληκτρολογήσετε θα επιστραφεί σε εσάς στην έξοδο. Εκτελέστε το πρόγραμαμ αρκετές φορές και δοκιμάστε να πληκτρολογήσετε διαφορετικά πράγματα κάθε φορά. Τώρα μπορείτε να φτιάξετε διαλογικά προγράμματα! Στο επόμενο πρόγραμμα, πληκτρολογήστε το όνομα σας και σαν απάντηση θα σας χαιρετήσει.

```
puts 'Hello there, and what\'s your name?'
name = gets
puts 'Your name is ' + name + '?  What a lovely name!'
puts 'Pleased to meet you, ' + name + '.  :)'
```

Δοκιμάστε το, εκτελέστε το, πληκτρολογήστε το όνομα σας και δείτε τι συμβαίνει:

```
Hello there, and what's your name?
Chris
Your name is Chris
?  What a lovely name!
Pleased to meet you, Chris
.  :)
```

Φαίνεται πως όταν πληκτρολογήσατε τα γράμματα C,h,r,i,s, και στη συνέχεια πατήσατε το E n t e r η μέθοδος gets, λαμβάνει όλα τα γράμματα που πληκτρολογήσατε, αλλά λαμβάνει και το Enter. Γι' αυτό προσέξτε πως κάθε φορά που εμφανίζεται η μεταβλητή name οποιοσδήπτοε χαρακτήρας ακολουθεί, εμφανίζεται στην επόμενη γραμμή. Ευτυχώς υπάρχει μια μέθοδος για αυτό το πράγμα: η chomp. Αγνοεί το πλήκτρο Enter και παραμένει στον τελευταίο χαρακτήρα του αλφαριθμητικού που πληκτρολογήσατε. Προσπαθήστε και πάλι το προηγούμενο πρόγραμμα, αλλά αφήστε τη μέθοδο chomp να σας βοηθήσει αυτή τη φορά:

```
puts 'Hello there, and what\'s your name?'
name = gets.chomp
puts 'Your name is ' + name + '?  What a lovely name!'
puts 'Pleased to meet you, ' + name + '.  :)'
```

```
Hello there, and what's your name?
Chris
Your name is Chris?  What a lovely name!
Pleased to meet you, Chris.  :)
```

Πολύ καλύτερα. Προσέξτε πως από τη στιγμή που η μεταβλητή name δείχνει προς τη μέθοδο gets.chomp δεν χρειάζεται καν να πούμε name.chomp,  η μεταβλητή name, ήδη είχε οικοιοποιηθεί τη μέθοδο chomp.

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

- Γράψτε ένα πρόγραμμα, το οποίο θα ζητά το όνομα, το επίθετο και το πατρώνυμο ενός ανθρώπου. Στη συνέχεια θα πρέπει να χαιρετάει τον άνθρωπο χρησιμοποιώντας όλα τα στοιχεία του.
- Γράψτε ένα πρόγραμμα, το οποίο θα ζητά τον αγαπημένο αριθμό του χρήστη. Το πρόγραμμα θα αυξάνει κατά ένα τον αριθμό που δώσατε και στη συνέχεια θα σας προτείνει τον καινούριο αριθμό ως μεγαλύτερο και καλύτερο αγαπημένο αριθμό.

Αφού τελειώσετε με αυτά τα δύο προγράμματα (και πιθανά άλλα που θα θέλατε να δοκιμάσετε), είναι ώρα να μάθουμε περισσότερα πράγματα σχετικά με τις μεθόδους.

## 5. Περισσότερα σχετικά με τις μεθόδους

Μέχρι τώρα, έχουμε δει μερικές διαφορετικές μεθόδους, όπως την puts και την gets (Δημιουργήστε μια λίστα με όλες τις διαφορετικές μεθόδους που έχουμε δει μέχρι τώρα. Θα πρέπει να εντοπίσετε δέκα, η απάντηση δίνεται παρακάτω). Ωστόσο δεν έχουμε πραγματικά αναλύσει σχετικά με το τι μέθοδοι είναι αυτές. Γνωρίζουμε τι κάνουν, αλλά δεν γνωρίζουμε τι είναι.

Να λοιπόν τι είναι: Πράγματα, τα οποία κάνουν διάφορα.

Αν τα αντικείμενα (όπως τα αλφαριθμητικά οι ακέραιοι και οι πραγματικοί) είναι τα ουσιαστικά στη γλώσσα Ruby, τότε οι μέθοδοι είναι κάτι σαν τα ρήματα. Ακριβώς όπως στα αγγλικά δεν μπορείτε να έχετε ένα ρήμα χωρίς ένα ουσιαστικό (υποκείμενο) να δρα με το ρήμα. Για Για παράδειγμα, το τικ τακ δεν είναι κάτι που απλά συμβαίνει, ένα ρολόι είναι αυτό που οφείλει να κάνει το τικ τακ. Μιλώντας λέμε “το ρολόι κάνει τικ τακ”. Στην Ruby θα λέγαμε clock.tick (υποθέτοντας βέβαια πως το ρολόι -clock-, θα είναι ένα αντικείμενο). Οι προγραμματιστές θα μπορούσαν να πούνε οτι: “καλούν την μέθοδο τικ τακ του ρολογιού” ή οτι “καλούν ένα τικ τακ στο ρολόι”.

Λοιπόν είναι σίγουρο οτι θυμάστε τις μεθόδους puts, gets και chomp, αφού μόλις τις μάθαμε πως λειτουργούν και τι σημαίνουν. Πιθανά κατανοήσατε και τις μεθόδους μετατροπής, .to_i, .to_f, .to_s. Παρόλα αυτά, έχουμε άλλα τέσσερα πραγματάκια και δεν είναι τίποτα άλλο παρά οι παλιοί γνωστοί μας αριθμητικοί τελεστές +, -, * και / .

Όπως λέχθηκε νωρίτερα κάθε ρήμα χρειάζεται ένα ουσιαστικό (ως υποκείμενο), έτσι και κάθε μέθοδος χρειάζεται ένα αντικείμενο. Είναι συνήθως εύκολο να εντοπιστεί πιο αντικείμενο χρησιμοποιεί μια μέθοδο, είναι αυτό το οποίο είναι γραμμένο ακριβώς πριν την τελεία έτσι στο παράδειγμα clock.tick το αντικείμενο είναι το clock ή στο παράδειγμα 101.to_s το αντικείμενο είναι ο αριθμός 101. Μερικές φορές όμως δεν είναι τόσο φανερό το αντικείμενο, όπως παράδειγμα στις αριθμητικές μεθόδους. Έτσι η πράξη 5 + 5 είναι ένας πιο σύντομος τρόπος να γραφεί το 5.+ 5 . Για παράδειγμα:

```
puts 'hello '.+ 'world'
puts (10.* 9).+ 9
```

```
hello world
99
```

Βέβαια δεν είναι πολύ εμφανίσιμο, αλλά και πολύ βολικό, γι' αυτό και σχεδόν ποτέ δεν θα το γράφουμε έτσι. Είναι όμως σημαντικό να κατανοήσετε τι ακριβώς συμβαίνει. Μάλιστα μερικές

φορές με τον παραπάνω τρόπος γραφής μπορεί να σας βγάλει την παρακάτω προειδοποίηση: warning: parenthesize argument(s) for future version. Το πρόγραμμα εκτελείται, καθώς ο κώδικας είναι σωστός, απλά το παραπάνω μήνυμα σας λέει οτι έχει πρόβλημα να καταλάβει τι εννοείται και να χρησιμοποιείτε περισσότερες παρενθέσεις στο μέλλον. Αυτό επίσης κάνει περισσότερο κατανοητό γιατί μπορούμε να κάνουμε την πράξη 'pig'*5 , αλλά όχι την πράξη 5*'pig'. Το 'pig'*5 σημαίνει πως το αντικείμενο pig πολλαπλασιάζεται, χρησιμοποιεί την μέθοδο του πολλαπλασιασμού, αλλά στην περίπτωση του 5*'pig' σημαίνει πως το αντικείμενο 5 πολλαπλασιάζεται, δηλαδή χρησιμοποιεί τη μέθοδο του πολλαπλασιασμού. Όμως το 'pig' ξέρει πως να κάνει 5 αντίγραφα του εαυτού του και να τα προσθέσει όλα μαζί. Αντίθετα το 5 θα έχει πολύ μεγαλύτερη δυσκολία να κάνει 'pig' αντίγραφα του εαυτού του και να τα προσθέσει όλα μαζί.

Φυσικά δεν ξεχάσαμε πως έχουμε ακόμα να εξηγήσουμε την puts και την gets. Που είναι τα αντικείμενα τους; Στην καθημερινή σας γλώσσα πολλές φορές μπορείτε να παραλείπετε το ουσιαστικό. Στην Ruby, αν γράψετε: puts 'to be or not to be', αυτό στην πραγματικότητα σημαίνει: self.puts 'to be or not to be' Τι είναι όμως η προσθήκη self; Είναι μια ειδική μεταβλητή, η οποία δείχνει σε οποιοδήποτε αντικείμενο “βρίσκεστε μέσα”. Βέβαια ακόμα δεν γνωρίζετε πως να βρίσκεστε μέσα σε ένα αντικείμενο, αλλά μέχρι να το μάθετε, θα είστε πάντα μέσα σε ένα μεγάλο αντικείμενο, το οποίο δεν είναι τίποτα άλλο παρά ολόκληρο το πρόγραμμα. Ευτυχώς για μας το ίδιο το πρόγραμμα έχει μερικές δικές του μεθόδους όπως η puts και η gets . Δείτε αυτό:

```
iCantBelieveIMadeAVariableNameThisLongJustToPointToA3 = 3
puts iCantBelieveIMadeAVariableNameThisLongJustToPointToA3
self.puts iCantBelieveIMadeAVariableNameThisLongJustToPointToA3
```

```
3
3
```

Το σημαντικό που πρέπει να θυμάστε από αυτό το σημείο είναι πως κάθε μέθοδος υλοποιείται από ένα αντικείμενο, ακόμα και αν δεν υπάρχει κάποια τελεία μπροστά από τη μέθοδο και το αντικείμενο να προηγείται. Αν το έχετε κατανοήσει αυτό, τότε μπορείτε να είστε ικανοποιημένοι.

**Φανταχτερές αλφαριθμητικές μέθοδοι**

Ας μάθουμε μερικές αστείες αλφαριθμητικές μεθόδους. Δεν είναι ανάγκη να τις θυμάστε όλες, μπορείτε απλά να επανέρχεστε σε αυτή τη σελίδα αν τις ξεχνάτε. Σκοπός είναι απλά να δείτε έστω ένα, ν μικρό μέρος από αυτά που μπορούν να κάνουν τα αλφαριθμητικά . Εξάλλου υπάρχουν τόσες σπουδαίες αναφορές στο διαδίκτυο, με όλες τις μεθόδους για αλφαριθμητικά, καταλογογραφημένες και με επεξηγήσεις (στο τέλος του εγχειριδίου θα δοθούν παραπομπές που να τις βρείτε). Πραγματικά δεν χρειάζεται να ξέρετε όλες τις αλφαριθμητικές μεθόδους. Είναι σαν να θέλετε να ξέρετε όλες τις λέξεις στο λεξικό της γλώσσας που μιλάτε. Μπορείτε να μιλήσετε άριστα τη γλώσσας σας, χωρίς να γνωρίζετε όλες τις λέξεις το λεξικό, και πραγματικά αυτός δεν είναι ο σκοπός ύπαρξης του λεξικού; να μην χρειάζεται να ξέρετε οτιδήποτε βρίσκεται σε αυτό;

Έτσι η πρώτη μας αλφαριθμητική μέθοδος είναι η reverse, η οποία δίνει ένα αλφαριθμητικό αντεστραμμένο:

```
var1 = 'stop'
var2 = 'stressed'
var3 = 'Can you pronounce this sentence backwards?'

puts var1.reverse
```

```
puts var2.reverse
puts var3.reverse
puts var1
puts var2
puts var3
```

```
pots
desserts
?sdrawkcab ecnetnes siht ecnuonorp uoy naC
stop
stressed
Can you pronounce this sentence backwards?
```

Όπως βλέπετε η reverse, δεν αντιστρέφει το αυθεντικό αλφαριθμητικό (δηλαδή αυτό που είναι αποθηκευμένο στη μεταβλητή). Απλά δημιουργεί μια νέα αντίστροφη εκδοχή από αυτό. Γι' αυτό η μεταβλητή var1, ακόμα περιέχει το 'stop', ακόμα και αφού καλέσαμε την reverse στη μεταβλητή var1.

Μια ακόμα μέθοδος για αλφαριθμητικά είναι η length, η οποία μας λέει τον αριθμό ων χαρακτήρων (περιλαμβανομένων των κενών) του αλφαριθμητικού:

```
puts 'What is your full name?'
name = gets.chomp
puts 'Did you know there are ' + name.length + ' characters in your name, ' + name + '?'
```

```
What is your full name?
Christopher David Pine
#<TypeError: can't convert Fixnum into String>
```

Ωχ! Κάτι πήγε στραβά και φαίνεται να συνέβη, λίγο μετά την γραμμή name = gets.chomp... Βλέπετε το πρόβλημα;? Δείτε αν μπορείτε να το εντοπίσετε.

Το πρόβλημα είναι με τη μέθοδο length: Η μέθοδος αυτή μας επιστρέφει έναν αριθμό, αλλά εμείς θέλουμε ένα αλφαριθμητικό. Αρκετά εύκολο, απλά θα προστεθεί ένα .to_s .

```
puts 'What is your full name?'
name = gets.chomp
puts 'Did you know there are ' + name.length.to_s + ' characters in your name, ' + name + '?'
```

```
What is your full name?
Christopher David Pine
Did you know there are 22 characters in your name, Christopher David Pine?
```

**Σημείωση:** Αυτός είναι ο αριθμός των χαρακτήρων στο όνομα μου και όχι ο αριθμός των γραμμάτων (μετρήστε τα). Δημιουργήστε ένα πρόγραμμα, το οποίο θα ζητά πρώτα το όνομα σας, στη συνέχεια το επίθετο και τέλος το πατρώνυμο, ξεχωριστά το καθένα και στη συνέχεια προσθέτει το μήκος των χαρακτήρων και εμφανίζει το συνολικό πλήθος των χαρακτήρων. Να έχετε υπόψιν πως πιθανά εφόσον γράψετε τα στοιχεία σας με ελληνικούς χαρακτήρες το πλήθος αυτών στο αποτέλεσμα θα είναι διπλάσιο από τους χαρακτήρες που πραγματικά πληκτρολογήσατε.

Τα καταφέρατε; ωραία, είναι ένα καλό πρόγραμμα που καταφέρατε να φτιάξετε με λίγα μόνο μαθήματα. Πολύ σύντομα θα εντυπωσιαστείτε με αυτά που θα μπορείτε να κάνετε.

Υπάρχουν επίσης, μέθοδοι, οι οποίες μετατρέπουν σε κεφαλαία η σε πεζά τα

γράμματα ( του αλφαριθμητικου σας . Η μέθοδος upcase αλλάζει κάθε πεζό γραμμα σε κεφαλαίο, και η μέθοδος downcase, αλλάζει κάθε κεφαλαίο γραμμα σε πεζό. Η μέθοδος swapcase πραγματοποιεί είτε την μία είτε την άλλη μετατροπή, ανάλογα με την αρχική κατάσταση των γραμμάτων και τέλος η μέθοδος capitalize λειτουργεί όπως και η downcase, εκτός από το οτι μετατρέπει τον πρώτο χαρακτήρα σε κεφαλαίο (αν είναι γράμμα φυσικά).

```
letters = 'aAbBcCdDeE'
puts letters.upcase
puts letters.downcase
puts letters.swapcase
puts letters.capitalize
puts ' a'.capitalize
puts letters
```

```
AABBCCDDEE
aabbccddee
AaBbCcDdEe
Aabbccddee
 a
aAbBcCdDeE
```

Πολύ ωραίο υλικό. Όπως μπορείτε να δείτε στην παρακάτω γραμμή: puts ' a'.capitalize, η μέθοδος capitalize μετατρέπει σε κεφαλαίο μόνο τον πρώτο χαρακτήρα και όχι το πρώτο γράμμα. . Στην περίπτωση αυτή ο πρώτος χαρακτήρας είναι το κενό. Επίσης όπως έχουμε ξαναδεί και σε άλλα παραδείγματα, μέσα από την κλήση όλων αυτών των διαδικασιών η μεταβλητή letters παραμένει αμετάβλητη. Όχι πως δεν υπάρχουν μέθοδοι που αλλάζουν το συσχετιζόμενο αντικείμενο, αλλά προς το παρόν δεν έχετε μάθει κάποια τέτοια μέθοδο και δεν θα μάθετε για λίγο ακόμα.

Οι τελευταίες από τις εντυπωσιακές μεθόδους, τις οποίες θα μάθετε, αφορούν την οπτική μορφοποίηση. Η πρώτη μέθοδος είναι η center, η οποία προσθέτει κενούς χαρακτήρες στην αρχή και το τέλος του αλφαριθμητικού για να το κεντράρει. Έτσι όπως ακριβώς πρέπει να προσδιορίσετε στην puts τι θέλετε να εμφανίσει (τυπώσει) και στο + τι ακριβώς θέλετε να προσθέσετε, έτσι πρέπει να πείτε στη μέθοδο center, πόσο πλάτος θέλετε να έχει το κεντραρισμένο αλφαριθμητικό σας. Έτσι αν θέλετε να κεντράρετε τους στίχους ενός ποιήματος, θα πρέπει να το κάνετε κάπως έτσι:

```
lineWidth = 50
puts(              'Old Mother Hubbard'.center(lineWidth))
puts(             'Sat in her cupboard'.center(lineWidth))
puts(       'Eating her curds an whey,'.center(lineWidth))
puts(        'When along came a spider'.center(lineWidth))
puts(       'Which sat down beside her'.center(lineWidth))
puts('And scared her poor shoe dog away.'.center(lineWidth))
```

```
            Old Mother Hubbard
           Sat in her cupboard
         Eating her curds an whey,
          When along came a spider
         Which sat down beside her
      And scared her poor shoe dog away.
```

Προσέξτε πως αλλάζει η τοποθέτηση των γραμμών του ποιήματος, με τη βοήθεια κενών

χαρακτήρων. Προσθέτοντας κενούς χαρακτήρες, προσμετρώνται και αυτοί στο αλφαριθμητικό μας και υπολογίζονται στο συνολικό μήκος του αλφαριθμητικού, ώστε να κρατηθούν οι ανάλογες αποστάσεις από τα άκρα. Επίσης προσέξτε πως αποθηκεύθηκε το πλάτος τς γραμμής στη μεταβλητή lineWidth. Αυτό έγινε για λόγους τεμπελιάς. Έτσι αν θέλετε αργότερα να αλλάξετε το πλάτος των γραμμών του ποιήματος, θα πρέπει να αλλάξετε μόνο την πρώτη γραμμή του προγράμματος, αντί να αλλάζετε κάθε γραμμή που "κεντράρει" έναν στίχο του ποιήματος. Σε ένα πολύ μεγάλο ποίημα, αυτή η επιλογή θα μπορούσε να σας γλιτώσει πολύ χρόνο. Αυτού του είδους η τεμπελιά, πραγματικά αποτελεί αρετή στον προγραμματισμό.

Σχετικά με το κεντράρισμα, πιθανά θα έχετε προσέξει οτι δεν είναι τόσο όμορφο σε σχέση με το τι θα μπορούσε να κάνει ένας επεξεργαστής κειμένου. Αν πραγματικά θέλετε τέλειο κεντράρισμα (και πιθανά και μια ωραιότερη γραμαμτοσειρά), τότε θα έπρεπε να χρησιμοποιήσετε έναν επεξεργαστή κειμένου. Η Ruby είναι ένα υπέροχο εργαλείο, αλλά όχι εργαλείο για κάθε δουλειά.

Οι άλλες δύο μέθοδοι μορφοποίησης αλφαριθμητικών είναι η ljust και η rjust, οι οποίες αντιπροσωπεύουν την αριστερή στοίχιση ( left justify) και τη δεξιά στοίχιση ( right justify).Οι δύο αυτές μέθοδοι είναι παρόμοιες με την center, εκτός από το οτι γεμίζουν το αλφαριθμητικό με κενά στην δεξιά και την αριστερή πλευρά αντίστοιχα. Δείτε και τις τρεις μεθόδους σε δράση:

```
lineWidth = 40
str = '--> text <--'
puts str.ljust  lineWidth
puts str.center lineWidth
puts str.rjust  lineWidth
puts str.ljust (lineWidth/2) + str.rjust (lineWidth/2)
```

```
--> text <--
            --> text <--
                        --> text <--
--> text <--                --> text <--
```

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

• Γράψτε ένα πρόγραμμα με θέμα Το θυμωμένο αφεντικό. Το πρόγραμμα θα σας ρωτάει με αγένεια

"Τι θέλεις;" Οτιδήποτε και να απαντήσετε το θυμωμένο αφεντικό θα πρέπει να σας το ξαναλέι ουρλιάζοντας και να σας απολύει. Για παράδειγμα αν πληκτρολογήσετε: "Θέλω αύξηση" θα πρέπει να σας φωνάζει: "Τι εννοείς -Θέλω αύξηση;- Απολύεσαι"

• Ορίστε και κάτι, για να εξασκηθείτε περισσότερο στις μεθόδους center, ljust και rjust: Γράψτε ένα πρόγραμμα το οποίο θα εμφανίζει έναν Πίνακα Περιεχομένων ο οποίος θα μοιάζει περίπου σαν τον παρακάτω:

```
            Table of Contents

Chapter 1:  Numbers                 page 1
Chapter 2:  Letters                 page 72
Chapter 3:  Variables               page 118
```

**Ανώτερα μαθηματικά**

*(Αυτή η ενότητα είναι προαιρετική. Προϋποθέτει ένα ικανοποιητικό βαθμό γνώσης των μαθηματικών. Αν δεν σας ενδιαφέρει μπορείτε να πάτε κατ' ευθείαν στον Έλεγχο Ροής (Κεφάλαιο 6), χωρίς κανένα πρόβλημα. Παρόλα αυτά μια γρήγορη ματιά στην ενότητα των* ***Τυχαίων Αριθμών*** πιθανά να σας φανεί πρακτική.

Δεν υπάρχουν τόσες πολλές αριθμητικές μέθοδοι όσες υπάρχουν για τα αλφαριθμητικά (παρόλα αυτά είναι εξίσου δύσκολο να τις θυμάται κάποιος όλες) Ας δούμε λοιπόν μερικές ακόμα αριθμητικές μεθόδους, μια γεννήτρια τυχαίων αριθμών και το αντικείμενο Math με τις τργωνομετρικές και υπερβατικές (μη αλγεβρικές) μεθόδους.

**Περισσότερη αριθμητική**

Οι άλλες δύο αριθμητικές μέθοδοι είναι η ** (ύψωση σε δύναμη-exponentiation) και % (υπόλοιπο διαίρεσης-modulus)Έτσι αν θέλετε να πείτε "πέντε στο τετράγωνο" στην Ruby θα έπρεπε να το γράψετε 5**2. Μπορείτε επίσης να χρησιμοποιήσετε δεκαδικούς σαν δύναμη, έτσι αν θέλετε την τετραγωνική ρίζα του 5 θα μπορούσατε να γράψετε 5**0.5 Η μέθοδος modulus σας δίνει το υπόλοιπο μιας διαίρεσης με έναν αριθμό. Έτσι για παράδειγμα αν διαιρέσετε το 7 με το 3 θα πάρετε πηλίκο 2 με υπόλοιπο 1. Δείτε το να δουλεύει σε ένα πρόγραμμα.

```
puts 5**2
puts 5**0.5
puts 7/3
puts 7%3
puts 365%7
```

```
25
2.23606797749979
2
1
1
```

Από την τελευταία γραμμή μαθαίνετε οτι ένα (μη δίσεκτο) έτος έχει έναν αριθμό εβδομάδων και επιπλέον μία μέρα. Έτσι αν τα γενέθλια σας είναι Τρίτη φέτος, την επόμενη χρονιά θα είναι Τετάρτη. Μπορείτε να χρησιμοποιήσετε και δεκαδικούς στην μέθοδο modulus. Βασικά δουλεύει με τον μόνο τρόπο που θα μπορούσε, αλλά πειραματιστείτε με αυτό.

Υπάρχει ακόμα μία μέθοδος που πρέπει να αναφερθεί εδώ πριν εξηγήσουμε την γεννήτρια τυχαίων αριθμών. Η μέθοδος abs, απλά επιστρέφει την απόλυτη τιμή ενός αριθμού:

```
puts((5-2).abs)
puts((2-5).abs)
```

```
3
3
```

**Τυχαίοι Αριθμοί**

Η Ruby περιλαμβάνει μία πολύ καλή γεννήτρια τυχαίων αριθμών. Η μέθοδος για να λάβετε έναν τυχαία επιλογμένο αριθμό είναι η rand. Αν καλέσετε έτσι απλά την rand, θα λάβετε έναν δεκαδικό μεγαλύτερο ή ίσο με 0.0 και μικρότερο από 1.0. Αν δώσετε στην rand ως παράμετρο έναν ακέραιο (πχ 5) τότε θα σας επιστρέψει έναν ακέραιο μεγαλύτερο ή ίσο από το 0 και μικρότερο από

το 5 (έτσι πέντε είναι οι τυχαίοι αριθμοί από το 0 έως και το 4)

Δείτε την rand σε δράση:

```
puts rand
puts rand
puts rand
puts(rand(100))
puts(rand(100))
puts(rand(100))
puts(rand(1))
puts(rand(1))
puts(rand(1))
puts(rand(99999999999999999999999999999999999999999999999999999999999))
puts('Ο μετεωρολόγος είπε οτι υπάρχει '+rand(101).to_s+'% πιθανότητα για βροχή,')
puts('αλλά δεν μπορείς ποτέ να εμπιστευτείς έναν μετεωρολόγο.')
```

```
0.866769322351658
0.155609260113273
0.208355946789083
61
46
92
0
0
0
22982477508131860231954108773887523861600693989518495699862
Ο μετεωρολόγος είπε οτι υπάρχει 47% πιθανότητα για βροχή,
αλλά δεν μπορείς ποτέ να εμπιστευτείς έναν μετεωρολόγο.
```

Σημειώστε οτι χρησιμοποίησα rand(101) για να πάρω πίσω αριθμούς από το 0 έως το 100 και πως η rand(1) επιστρέφει πάντα 0. Να θυμάστε πως το εύρος των πιθανών επιστρεφόμενων τιμών είναι το μεγαλύτερο λάθος που κάνουνε οι άνθρωποι με την rand. Ακόμα και οι επαγγελματίες προγραμματιστές, ακόμα και σε ολοκληρωμένα προϊόντα που μπορείτε να αγοράσετε σε καταστήματα. Έχουν υπάρξει περιπτώσεις cd players τα οποία αν τα έβαζες σε λειτουργία τυχαίας επιλογής κομματιού, θα έπαιζαν όλα τα κομμάτια εκτός από το τελευταίο. Σκεφτείτε τι θα συνέβαινε αν ένα cd είχε μόνο ένα τραγούδι.

Μερικές φορές μπορεί να θέλετε η rand να σας επιστρέψει τους ίδιους τυχαίους αριθμούς στην ίδια σειρά σε δύο διαφορετικές εκτελέσεις του προγράμματος σας. (για παράδειγμα αν χρησιμοποιείτε την γεννήτρια τυχαίων αριθμών για να δημιουργήσετε έναν τυχαία "γεννημένο" κόσμο για ένα παιχνίδι υπολογιστή. Αν δημιουργηθεί ένας κόσμος που σας αρέσει και θέλετε να ξαναπαίξετε στον ίδιο ή να τον στείλετε σε έναν φίλο. Για να το κάνετε αυτό χρειάζεται να *ορίσετε τον "σπόρο" που θα παράγει τους αριθμούς*, κάτι το οποίο μπορείτε να το κάνετε με την srand. Δείτε το παρακάτω παράδειγμα:

```
srand 1776
puts(rand(100))
puts(rand(100))
puts(rand(100))
puts(rand(100))
puts(rand(100))
```

```
puts ''
srand 1776
puts(rand(100))
puts(rand(100))
puts(rand(100))
puts(rand(100))
puts(rand(100))
```

```
24
35
36
58
70

24
35
36
58
70
```

Θα παίρνετε τα ίδια αποτελέσματα κάθε φορά που θα δίνετε τον ίδιο αριθμό (σπόρο) στην srand. Αν θέλετε να πάρετε πάλι διαφορετικούς αριθμούς (όπως θα συνέβαινε αν δεν είχατε χρησιμοποιήσει ποτέ την srand) τότε χρησιμοποιήστε την srand 0. Καλώντας με αυτόν τον τρόπο την srand, στην ουσία την τροφοδοτείτε την με έναν αριθμό που παράγεται από την τρέχουσα ώρα του υπολογιστή σας μέχρι επίπεδο χιλιοστού του δευτερολέπτου.

**Το αντικείμενο Math**

Τέλος ώρα να μάθετε το αντικείμενο Math.

```
puts(Math::PI)
puts(Math::E)
puts(Math.cos(Math::PI/3))
puts(Math.tan(Math::PI/4))
puts(Math.log(Math::E**2))
puts((1 + Math.sqrt(5))/2)
```

```
3.14159265358979
2.71828182845905
0.5
1.0
2.0
1.61803398874989
```

Το πρώτο πράγμα που πιθανά προσέξατε είναι πιθανά το σύμβολο ::. Εξηγώντας τον σκοπό του τελεστή αυτού είναι κάτι πέρα από τον σκοπό του εγχειριδίου, ψστόσο μπορεί να χρησιμοποιήσετε για παράδειγμα την έκφραση Math:PI για να έχετε το αποτέλεσμα που προσδοκάτε να έχετε (δηλαδή να σας επιστρέφει τον αριθμό π)

Όπως είδατε από τα παραδείγματα το αντικείμενο Math εχει όλα εκείνα τα στοιχεία που θα προσδοκούσατε από έναν αξιοπρεπή επιστημονικό αριθμητικό υπολογιστή. Και φυσικά όπως πάντα οι δεκαδικοί αριθμοί είναι πραγματικά πιο πιθανό να είναι στις σωστές απαντήσεις.

Ώρα να ασχοληθείτε με την “ροή”.

## 6. Έλεγχος ροής.

Σε αυτό το σημείο όλα όσα μάθατε έρχονται μαζί και “συνεργάζονται”. Αν και αυτό το κεφάλαιο είναι μικρότερο και ευκολότερο από το κεφάλαιο των μεθόδων, θα εμφανίσει μπροστά στα μάτια σας έναν ολόκληρο κόσμο προγραμματιστικών δυνατοτήτων. Μετά από αυτό το κεφάλαιο θα είστε ικανοί να γράψετε πραγματικά διαλογικά προγράμματα. Σε προηγούμενες ενότητες, δημιουργήσατε προγράμματα, τα οποία εμφάνιζαν διάφορα πράγματα, εξαρτώμενα από τα δεδομένα που εισάγατε από το πληκτρολόγιο. Μετά από αυτό το κεφάλαιο όμως, πραγματικά θα κάνουν διαφορετικά πράγματα επίσης. Πριν όμως κάνετε αυτό, πρέπει πρώτα να είστε ικανοί να συγκρίνετε αντικείμενα στα προγράμματα σας. Χρειάζεστε τις.....

### Μεθόδους σύγκρισης

Βιαστείτε λιγάκι σε αυτό το κομμάτι για να πάτε γρήγορα στην επόμενη ενότητα την, **διακλάδωση,** όπου όλα τα ωραία πράγματα συμβαίνουν. Έτσι αν θέλετε να διαπιστώσετε αν ένα αντικείμενο είναι μεγαλύτερο ή μικρότερο από ένα άλλο χρησιμοποιείστε τις μεθόδους > και <, όπως παρακάτω:

```
puts 1 > 2
puts 1 < 2
```

```
false
true
```

Παρόμοια, μπορείτε να βρείτε αν ένα αντικείμενο είναι μεγαλύτερο ή ίσο και μικρότερη ή ίσο από ένα άλλο με τις μεθόδους >= και <=

```
puts 5 >= 5
puts 5 <= 4
```

```
true
false
```

Τέλος, μπορείτε να διαπιστώσετε αν δύο αντικείμενα είναι ίσα ή όχι χρησιμοποιώντας τη μέθοδο == (η οποία σημαίνει “είναι αυτά ίσα;") και != (η οποία σημαίνει "είναι αυτά διαφορετικά;"). Είναι πολύ σημαντικό να μην συγχέετε το = με το ==. Το = λέει σε μια μεταβλητή να “δείξει” προς ένα αντικείμενο (εκχώρηση τιμής στην μεταβλητή) και το == σας θέτει το ερώτημα: “Είναι αυτά τα δύο αντικείμενα ίσα;”

```
puts 1 == 1
puts 2 != 1
```

```
true
true
```

Φυσικά, μπορείτε να συγκρίνετε αλφαριθμητικά επίσης. Όταν τα αλφαριθμητκά συγκρίνονται, γίνεται σύγκριση της λεξικογραφικής σειράς, η οποία βασικά σημαίνει η σειρά τους στο λεξικό. Έτσι, αφού η λέξη cat, είναι πριν τη λέξη dog στο λεξικό, τότε:

```
puts 'cat' < 'dog'
```

```
true
```

Ωστόσο υπάρχει μια παγίδα: Οι υπολογιστές συνήθως θεωρούν πως τα κεφαλαία γράμματα είναι πρωθύστερα των πεζών γραμμάτων(για παράδειγμα στις γραμαμτοσειρές, αποθηκεύουν πρώτα όλα τα κεφαλαία και μετά όλα τα πεζά). Αυτό σημαίνει πως ο υπολογιστής θα θεωρήσει πως η λέξη “Zoo” είναι πρωθύστερη (άρα και “μικρότερη”) από τη λέξη “ant”. Ετσι αν θέλετε να διαπιστώσετε ποια λέξη προηγείται σε ένα πραγματικό λεξικό βεβαιωθείτε πως θα χρησιμοποιήσετε, είτε κεφαλαία, είτε πεζά, είτε το πρώτο γράμμα κεφαλαίο, αλλά και για τις δυο λέξεις ακριβώς τον ίδιο τρόπο γραφής πριν προχωρήσετε στην σύγκριση τους.

Μια τελευταία επισήμανση πριν την **Διακλάδωση**: Οι μέθοδοι σύγκρισης δεν μας δίνουνε ως αποτέλεσμα τα αλφαριθμητικά 'true' και 'false'; αλλά τα ειδικά αντικείμενα true και false. (Φυσικά, true.to_s επιστρέφει 'true', κάτι που εξηγεί γιατί η puts τύπωσε 'true'.) Οι τιμές true και false χρησιμοποιούνται διαρκώς στη.....

**Διακλάδωση (Επιλογή)**

Η διακλάδωση είναι μια απλή ιδέα, αλλά πολύ δυνατή. Στην πραγματικότητα είναι τόσο απλή, που στοιχηματίζω οτι δεν χρειάζεται να σας την εξηγήσουμε. Απλά δείτε την:

```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if name == 'Chris'
  puts 'What a lovely name!'
end
```

```
Hello, what's your name?
Chris
Hello, Chris.
What a lovely name!
```

Αν όμως βάλετε διαφορετικό όνομα...

```
Hello, what's your name?
Chewbacca
Hello, Chewbacca.
```

Αυτό ακριβώς που είδατε είναι η διακλάδωση. Αν η σχέση που ακολουθεί την If, είναι αλήθεια τότε εκτελούνται οι εντολές ανάμεσα στην if και το end. Αν η σχέση που ακολουθεί την If είναι ψευδής, τότε οι εντολές δεν εκτελούνται. Απλά και ξάστερα.

Προσέξτε οτι ο κώδικας ανάμεσα στο If kai to End είναι γραμμένος με εσοχή.Με αυτό τον τρόπο είναι ευκολότερο να εντοπίζετε την ισχύ των διάφορων επιλογών της διακλάδωσης. Σχεδόν όλοι οι προγραμματιστές ακολουθούν, αυτόν τον τρόπο γραφής, άσχετα από τη γλώσσα προγραμματισμού που χρησιμοποιούν. Ίσως αυτή η τακτική δεν φαίνεται πολύ χρήσιμη σε αυτό το απλό παράδειγμα, αλλά όταν τα πράγματα γίνονται πιο πολύπλοκα, τότε ξ ωφέλεια είναι πολύ μεγάλη.

Συχνά, θα θέλαμε ένα πρόγραμμα να εκτελεί κάποιες εντολές αν μια έκφραση είναι αληθής, και άλλες εντολές αν η έκφραση είναι ψευδής. Γι' αυτό υπάρχει το else:

```
puts 'I am a fortune-teller.  Tell me your name:'
name = gets.chomp
if name == 'Chris'
  puts 'I see great things in your future.'
else
  puts 'Your future is... Oh my!  Look at the time!'
  puts 'I really have to go, sorry!'
end
```

```
I am a fortune-teller.  Tell me your name:
Chris
I see great things in your future.
```

Τώρα δοκιμάστε το ίδιο παράδειγμα, αλλά πληκτρολογήστε άλλο όνομα:

```
I am a fortune-teller.  Tell me your name:
Ringo
Your future is... Oh my!  Look at the time!
I really have to go, sorry!
```

Η διακλάδωση (επιλογή) είναι σαν μια διχάλα στον κώδικα: Παίρνουμε το ένα μονοπάτι για τους ανθρώπους που ισχύει name==Chris ή αλλιώς παίρνουμε το άλλο μονοπάτι;

Και ακριβώς όπως στα κλαδιά ενός δέντρου μπορείτε να έχετε κλαδία που και τα ίδια γεννούνε άλλα κλαδιά:

```
puts 'Hello, and welcome to 7th grade English.'
puts 'My name is Mrs. Gabbard.  And your name is...?'
name = gets.chomp

if name == name.capitalize
  puts 'Please take a seat, ' + name + '.'
else
  puts name + '?  You mean ' + name.capitalize + ', right?'
  puts 'Don\'t you even know how to spell your name??'
  reply = gets.chomp

  if reply.downcase == 'yes'
    puts 'Hmmph!  Well, sit down!'
  else
    puts 'GET OUT!!'
  end
end
```

```
Hello, and welcome to 7th grade English.
My name is Mrs. Gabbard.  And your name is...?
chris
chris?  You mean Chris, right?
Don't you even know how to spell your name??
yes
Hmmph!  Well, sit down!
```

Εντάξει το πρώτο γράμμα κεφαλαίο...

```
Hello, and welcome TNG 7th grade English.
My name is Mrs. Gabbard.  And your name is...?
Chris
Please take a seat, Chris.
```

Μερικές φορές ίσως μπερδευτείτε προσπαθώντας να κατανοήσετε, που ακριβώς ανήκει το κάθε if και το κάθε else με το αντίστοιχο end. Μια καλή τεχνική είναι να γρράφετε το end, μαζί με το αντίστοιχο if . Αν λοιπόν γράφατε το παραπάνω πρόγραμμα θα σας συμβούλευα να ξεκινούσατε κάπως έτσι :

```
puts 'Hello, and welcome to 7th grade English.'
puts 'My name is Mrs. Gabbard.  And your name is...?'
name = gets.chomp

if name == name.capitalize
else
end
```

Μετά καλό θα ήταν να συμπληρώνατε με σχόλια, δηλαδή κείμενο που ο υπολογιστής θα αγνοήσει:

```
puts 'Hello, and welcome to 7th grade English.'
puts 'My name is Mrs. Gabbard.  And your name is...?'
name = gets.chomp

if name == name.capitalize
  #  She's civil.
else
  #  She gets mad.
end
```

Οτιδήποτε μετά από μια δίεση (#) θεωρείται ως σχόλιο (εκτός βέβαια αν γράψετε τη δίεση μέσα σε ένα αλφαριθμητικό). Στη συνέχεια, αντικαταστείστε τα σχόλια με κώδικα που δουλεύει κανονικά. Ορισμένοι προγραμματιστές, προτιμούν να αφήνουν τα σχόλια. Άλλοι όπως εγώ (Chris Pine) νομίζουν πως ο καλογραμμένος κώδικας, μιλάει για τον εαυτό του. Βέβαια η ύπραξη ή όχι σχολίων σχετίζεται και με την εξοικείωση με την γλώσσα προγραμματισμού. Όσο πιο καλά μαθαίνω την Ruby, τόσο λιγότερο χρησιμοποιώ τα σχόλια. Πραγματικά τις περισσότερες φορές θεωρώ οτι μου αποσπούν την προσοχή τα σχόλια. Είναι βέβαια μαι προσωπική επιλογή. Εσείς θα βρείτε το προσωπικό σας στυλ. Έτσι το τελικό στάδιο του κώδικα είναι το παρακάτω:

```
puts 'Hello, and welcome to 7th grade English.'
puts 'My name is Mrs. Gabbard.  And your name is...?'
name = gets.chomp

if name == name.capitalize
  puts 'Please take a seat, ' + name + '.'
else
  puts name + '?  You mean ' + name.capitalize + ', right?'
  puts 'Don\'t you even know how to spell your name??'
  reply = gets.chomp
```

```
 if reply.downcase == 'yes'
 else
 end
end
```

Και πάλι, γράψτε το if, το else και το end την ίδια στιγμή. Βοηθάει να κρατάτε τα ίχνη του που βρίσκεστε κάθε στιγμή στον κώδικα. Να ξέρετε για ποιο if ισχύει το else και το end. Επίσης, βοηθάει στο να φαίνετια η δουλειά που έχετε να κάνετε, ευκολότερη, επειδή μπορείτε να επικεντρώσετε, σε ένα μικρό τμήμα, όπως να συμπληρώσετε τον κώδικα ανάμεσα στο if και το else. Ένα άλλο όφελος από αυτή την τακτική είναι πως ο υπολογιστής καταλαβαίνει το πρόγραμμα σε οποιοδήποτε στάδιο. Δηλαδή οποιαδήποτε από τις ανολοκλήρωτες εκδόσεις του προγράμματος, είδατε προηγουμένως θα μπορούσε να εκτελεστεί κανονικά . Δεν ήταν ολοκληρωμένα, αλλά ήταν λειτουργικά προγράμματα (δηλαδή προγράμματα που "δούλευαν"). Με αυτό τον τρόπο θα μπορούσατε να ελέγξετε το πρόγραμμα τη στιγμή που το γράφετε και να βλέπετε διαρκώς το παραγόμενο αποτέλεσμα, αλλά και που χρειάζεται ακόμα δουλειά να γίνει. Όταν περνάει ένα πρόγραμμα όλους αυτούς τους ελέγχους, τότε είστε σίγουρος πως αυτό που έπρεπε να κάνετε το κάνατε.

Αυτές οι συμβουλές, θα σας βοηθήσουν να γράψετε προγράμματα με διακλάδωση, αλλά επίσης θα σας βοηθήσουν και με τον άλλο κύριο τύπο του ελέγχου ροής:

**Βρόχος**

Συχνά θα θέλετε, ο υπολογιστής σας να κάνει τα ίδια πράγματα ξανά και ξανά, τελικά αυτό είναι, στο οποίο υποτίθεται πως οι υπολογιστές είναι καλοί

Όταν λέτε στον υπολογιστή σας να συνεχίσει να επαναλαμβάνει κάτι, πρέπει επίσης να του πείτε πότε να σταματήσει την επανάληψη. Οι υπολογιστές ποτέ δεν βαριούνται, έτσι αν δεν τους πείτε να σταματήσουν, δεν θα το κάνουν και θα έχουμε μια επανάληψη δίχως τέλος. Για να το αποφύγετε αυτό, λέτε στον υπολογιστή να επαναλαμβάνει κάποιο τμήμα του προγράμματος όσο κάποι συνθήκη είναι αληθής (while). Αυτή η μέθοδος δουλεύει παρόμοια με την if:

```
command = ''

while command != 'bye'
 puts command
 command = gets.chomp
end

puts 'Come again soon!'
```

```
Hello?
Hello?
Hi!
Hi!
Very nice to meet you.
Very nice to meet you.
Oh... how sweet!
Oh... how sweet!
bye
Come again soon!
```

Και αυτό είναι ένας βρόχος. (Ισως προσέξατε την πρώτη κενή γραμμή στην αρχή της εξόδου; είναι από την πρώτη puts, πριν την πρώτη gets. Πως θα μπορούσατε να αλλάξετε το πρόγραμμα για να ξεφορτωθείτε, αυτή την πρώτη γραμμή; Δοκιμάστε, πειραματιστείτε!! Δούλεψε ακριβώς όπως το παραπάνω πρόγραμμα, εκτός από την πρώτη κενή γραμμή;)

Οι βρόχοι , σας επιτρέπουν να κάνετε όλων των ειδών τα ενδιαφέροντα πράγματα, όπως είμαι σίγουρος οτι μπορείτε να φανταστείτε. Παρόλα αυτά, μπορούν επίσης να δημιουργήσουν προβλήματα, αν κάνετε ένα λάθος. Τι θα γίνει αν ο υπολογιστής ας εγκλωβιστεί σε έναν ατέρμωνα βρόχο; Αν νομίζετε πως κάτι τέτοιο συνέβη,κρατήστε πατημένο το πλήκτρο Ctrl και πατήστε το C

Προτού ξεκινήσουμε να παίζουμε, με τους βρόχους, ας μάθουμε μερικά πράγματα για να κάνουμε τη δουλειά μας ευκολότερη .

**Ένα μικρο κομμάτι "λογικής"**

Ας ρίξουμε ξανά μια ματιά στο πρώτο μας πρόγραμμα διακλάδωσης. Αν η γυναίκας σας ερχόταν στο σπίτι, έβλεπε το πρόγραμμα, το εκτελούσε και δεν της έλεγε τι ωραίο όνομα που έχει. Φυσικά δεν θα θέλατε να την πληγώσετε (ή να κοιμηθείτε στον καναπέ). Οπότε ας το ξαναγράψουμε:

```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if name == 'Chris'
  puts 'What a lovely name!'
else
  if name == 'Katy'
    puts 'What a lovely name!'
  end
end
```

```
Hello, what's your name?
Katy
Hello, Katy.
What a lovely name!
```

Λοιπόν το παραπάνω πρόγραμμα δουλεύει, αλλά δεν είναι ένα καλό πρόγραμμα. Γιατί όχι; Ο καλύτερος κανόνας που έμαθα στον προγραμματισμό είναι να μην επαναλαμβάνομαι .Ειλικρινά θα μπορούσα να είχα γράψει ένα μικρό βιβλίο, σχετικά με την αξία του παραπάνω κανόνα. Στην περίπτωση μας επαναλάβαμε την εξής γραμμή: puts 'What a lovely name!'. Γιατί είναι αυτό τόσο σημαντικό; Σκεφτείτε τι θα συνέβαινε αν έκανα ένα λάθος όταν ξαναέγραφα την ίδια γραμή; Τι θα συνέβαινε αν ήθελα να αλλάξω τη λέξη 'lovely' σε 'beautiful' και στις δυο γραμές; Δεν είναι απλώς θέμα τεμπελιάς. Αλλά, όταν θέλω το πρόγραμμα να κάνει το ίδιο πράγμα όταν πληκτρολογούμε 'Chris' ή 'Katy', τότε θα έπρεπε πραγματικά να κάνει το ίδιο πράγμα :

```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if (name == 'Chris' or name == 'Katy')
  puts 'What a lovely name!'
end
```

```
Hello, what's your name?
```

```
Katy
Hello, Katy.
What a lovely name!
```

Πολύ καλύτερα από πριν, δεν συμφωνείτε; Για να δουλέψει το πρόγραμμα σωστά χρησιμοποίησα τον λογικό τελεστή or (ή). Οι υπόλοιποι λογικοί τελεστές είναι το and (και) και το not (όχι) . Είναι καλύτερα να χρησιμοποιείτε παρενθέσεις, όταν εργάζεστε με τους λογικούς τελεστές. Ας δούμε πως δουλεύουνε αυτοί:

```
iAmChris  = true
iAmPurple = false
iLikeFood = true
iEatRocks = false

puts (iAmChris  and iLikeFood)
puts (iLikeFood and iEatRocks)
puts (iAmPurple and iLikeFood)
puts (iAmPurple and iEatRocks)
puts
puts (iAmChris  or iLikeFood)
puts (iLikeFood or iEatRocks)
puts (iAmPurple or iLikeFood)
puts (iAmPurple or iEatRocks)
puts
puts (not iAmPurple)
puts (not iAmChris )
```

```
true
false
false
false

true
true
true
false

true
false
```

Ο μόνος τελεστής, που θα μπορούσε να σας ξεγελάσει είναι το or (ή). Συνήθως χρησιμοποιούμε την διάζευξη (or-ή), εννοώντας “είτε το ένα είτε το άλλο, αλλά όχι και τα δύο”. Για παράδειγμα αν σας πούνε: “Για επιδόρπιο μπορείτε να έχετε πίττα ή κέικ” δεν εννοούνε οτι μπορείτε να φάτε και από τα δύο. Από την άλλη μεριά, ένας υπολογιστής, όταν χρησιμοποιεί τη διάζευξη (or-ή) εννοεί “είτε το ένα είτε το άλλο είτε και τα δύο”. Εναλλακτικό τρόπος διατύπωσης του παραπάνω θα μπορούσε να είναι “τουλάχιστον ένα από αυτά να είναι αλήθεια” .

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

• *"99 bottles of beer on the wall..."* Γράψτε ένα πρόγραμμα, το οποίο θα εμφανίζει τους στίχους από το κλασσικό τραγούδι: "99 Bottles of Beer on the Wall.". Φυσικά θα χρησιμοποιήσετε βρόχο

(επανάληψη). Τους στίχους θα τους βρείτε: http://99-bottles-of-beer.net/lyrics.html

• Γράψτε ρο πρόγραμμα της "Κουφής γιαγιάς".Οτιδήποτε και αν λέτε στην γιαγιά (δηλαδή οτιδήποτε και αν πληκτρολογείτε), η γιαγιά θα πρέπει να σας απαντά: ΕΙΠΕΣ ΤΙΠΟΤΑ ΠΑΛΗΚΑΡΙ ΜΟΥ; εκτός και α το φωνάξετε (δηλαδή να πληκτρολογήσετε όλους τους χαρακτήρες κεφαλαίους). Αν φωνάξετε η γιαγιά μπορεί να σας ακούσει (ή τουλάχιστον έτσι νομίζει) και απαντάει φωνάζοντας: ΟΧΙ, ΟΧΙ ΑΠΟ ΤΟ 1938! Για να κάνετε το πρόγραμμα σας, πιο ρεαλιστικό , βάλτε τη γιαγιά να φωνάζει ένα διαφορετικό έτος κάθε φορά, ίσως ένα τυχαίο έτος μεταξύ του 1930 και του 1950.(Αυτό το κομμάτι με τα τυχαία έτη, είναι προαιρετικό και είναι πιο εύκολο για εσάς αν διαβάσετε την ενότητα για την γεννήτρια τυχαίων αριθμών της Ruby στο τέλος του κεφαλαίου των μεθόδων). Δεν επιτρέπεται να σταματήσετε να μιλάτε στη γιαγιά πριν τις φωνάξετε: "ΑΝΤΙΟ".
**Σημ.:** *Μην ξεχνάτε την* chomp*! Το* 'ΑΝΤΙΟ' μαζί *με Enter δεν ειναι το ίδιο με το* 'ΑΝΤΙΟ' *χωρίς αυτό.*
**Σημ. 2:** *Σκεφτείτε, ποια τμήματα του προγράμματος σας θα έπρεπε να συμβαίνουν ξανά και ξανά. Όλα αυτά θα έπρεπε να βρίσκονται μέσα στο βρόχο while.*

• Επεκτείνετε το πρόγραμμα της "Κουφής γιαγιάς": Τι θα γίνει αν η γιαγιά δεν θέλει να φύγετε; όταν της φωνάζετε 'ΑΝΤΙΟ', θα μπορούσε να προσποιηθεί, οτι δεν σας ακούει. Αλλάξτε το προηγούμενο πρόγραμμα σας, έτσι ώστε να πρέπει να φωνάξετε 'ΑΝΤΙΟ' τρεις φορές στη σειρά. Ελέγξτε το πρόγραμμα σας οτι δουλεύει σωστά. Αν φωνάζετε τρεις φορές 'ΑΝΤΙΟ' αλλά όχι στη σειρά, θα πρέπει να συνεχίζετε να μιλάτε στη γιαγιά.

• Δίσεκτα έτη. Γράψτε ένα πρόγραμμα, το οποίο θα σας ρωτά ένα έτος αρχής και ένα έτος τέλους και στη συνέχεια θα εμφανίζει όλα τα δίσεκτα έτη μεταξύ τους (περιλαμβανομένων των ετών που πληκτρολογήσατε, αν αυτά είναι δίσεκτα) Δίσεκτα είναι τα έτη που διαιρούνται ακριβώς με το τέσσερα (όπως το 1984 και το 2004). Παρόλα αυτά, έτη που διαιρούνται με το 100 δεν είναι δίσεκτα (όπως το 1800 και το 1900), εκτός αν είναι διαιρέσιμα με το 400 (όπως το 1600 και το 2000, τα οποία είναι πράγματι δίσεκτα έτη) .

Όταν τελειώσετε, όλα αυτά, κάντε ένα διάλειμμα. Έχετε ήδη μάθει πάρα πολλά. Συγχαρητήρια. Εκπλήσσεστε από τα πράγματα που είστε ικανοί να πείτε στον υπολογιστή να κάνει; Μερικά κεφάλαια ακόμα και θα είστε ικανοί να φτιάξετε οποιοδήποτε πρόγραμμα. Σοβαρά. Απλά σκεφτείτε πόσα πράγματα μπορείτε να κάνετε τώρα, που δεν μπορούσατε, χωρίς τους βρόχους και τις διακλαδώσεις. Τώρα ας μάθουμε ένα νέο είδος αντικειμένου. Ένα αντικείμενο που μας επιτρέπει να αποθηκεύουμε καταλόγους άλλων αντικειμένων: Τους πίνακες (arrays).

## 7. Πίνακες και επαναλήπτες

Ας γράψουμε ένα πρόγραμμα, το οποίο θα σας ζητά να πληκτρολογήσετε όσες λέξεις θέλετε (μία λέξη σε κάθε γραμμή, συνεχόμενα, μέχρι να πατήσετε το πλήκτρο Enter σε μία άδεια γραμμή), και το οποίο μετά επαναλαμβάνει τις λέξεις (τις εμφανίζει δηλαδή) σε αλφαβητικά σειρά.

Με αυτά που έχετε μάθει μέχρι τώρα, δεν είναι εφικτό να το κάνετε αυτό. Χρειάζεστε έναν τρόπο να αποθηκεύσετε μια άγνωστη ποσότητα λέξεων και κρατάτε "σημάδια" για όλες αυτές, έτσι ώστε να μην αναμιγνύονται. Χρειάζεται δηλαδή να τις βάλετε σε ένα είδος καταλόγου. Χρειάζεστε με άλλα λόγια τους πίνακες.

Ένας πίνακας είναι απλά ένας κατάλογος στον υπολογιστή σας. Κάθε θέση στον κατάλογο συμπεριφέρεται σαν μια μεταβλητή: μπορείτε να δείτε, σε τι αντικείμενο δείχνει (τι έχει αποθηκευμένο δηλαδή) κάθε θέση, και μπορείτε να την κάνετε να δείχνει σε διαφορετικό αντικείμενο (να αλλάξετε την τιμή που έχετε αποθηκεύσει). Ας δούμε μερικούς πίνακες:

```
[]
[5]
['Hello', 'Goodbye']

flavor = 'vanilla'        #  Αυτό δεν είναι πίνακας φυσικά...
[89.9, flavor, [true, false]]  # Αυτό όμως είναι...
```

Έτσι αρχικά έχουμε έναν άδειο πίνακα, στη συνέχεια έναν πίνακα που περιέχει απλώς έναν αριθμό και μετά έναν πίνακα, που περιέχει δύο αλφαριθμητικά. Μετά έχουμε μια απλά εκχώρηση τιμής σε μεταβλητή και τέλος έχουμε έναν πίνακα που περιέχει τρία αντικείμενα, εκ των οποίων το τελευταίο είναι ο πίνακας [true, false]. Θυμηθείτε οι μεταβλητές δεν είναι αντικείμενα έτσι ο τελευταίος μας πίνακας, στην πραγματικότητα δείχνει σε έναν δεκαδικό, ένα αλφαριθμητικό και έναν πίνακα. Ακόμα και αν θέσουμε τη μεταβλητή flavor να δείχνει σε κάτι άλλο αυτό δεν θα άλλαζε τον πίνακα. Για να βρίσκουμε ποιο εύκολα το κάθε συγκεκριμένο αντικείμενο σε έναν πίνακα, σε κάθε θέση του έχει δοθεί ένας αύξων αριθμός ως δείκτης. Οι προγραμματιστές (και παρεμπιπτόντως και οι περισσότεροι μαθηματικοί) ξεκινούν την αρίθμηση από το μηδέν, έτσι η πρώτη θέση στον πίνακα είναι η θέση 0. Δίνεται ένα παράδειγμα, πως θα έπρεπε να αναφερθούμε στα αντικείμενα ενός πίνακα:

```
names = ['Ada', 'Belle', 'Chris']
puts names
puts names[0]
puts names[1]
puts names[2]
puts names[3]  #  This is out of range.
```

```
Ada
Belle
Chris
Ada
Belle
Chris
nil
```

Προσέξτε πως η γραμμή: puts names τυπώνει όλα τα ονόματα που υπάρχουν στον πίνακα names. Στη συνέχεια η χρήση της: puts names[0] τυπώνει το πρώτο όνομα του πίνακα και η puts names[1] τυπώνει το δεύτερο...Ίσως σας μπερδεύουν αυτά, ωστόσο θα πρέπει να τα συνηθίσετε. Ίσως να βοηθούσε αν ξεκινούσατε να σκέφτεστε και να μετράτε από το μηδέν και να σταματήσετε να χρησιμοποιείτε λέξεις όπως "πρώτο" και "δεύτερο". .

Τέλος δοκιμάσαμε την εντολή: puts names[3], απλώς για να δούμε τι θα συμβεί. Περιμένατε κάποιο μήνυμα λάθους; Μερικές φορές όταν θέτετε ένα ερώτημα στον υπολογιστή, το ερώτημα σας δεν έχει κανένα νόημα για τον υπολογιστή. Τότε ο υπολογιστής σας επιστρέφει ένα μήνυμα λάθους. Μερικές φορές, όμως, μπορεί να θέσετε ένα ερώτημα και η απάντηση είναι τίποτα. Τι υπάρχει στη θέση τρία του πίνακα μας; Τίποτα. Ποια είναι η τιμή στην θέση name[3]? nil στην γλώσσα της Ruby σημαίνει τίποτα. Το nil είναι ένα ειδικό αντικείμενο, το οποίο βασικά σημαίνει "κανένα άλλο αντικείμενο". Δηλαδή όχι κάποιο από τα υπαρκτά αντικείμενα. Αν όλη αυτή η η αρίθμηση των θέσεων των πινάκων σας φοβίζει, δεν πρέπει. Συχνά μπορούμε να την αποφύγουμε τελείως, χρησιμοποιώντας διάφορες μεθόδους πινάκων, όπως αυτή:

**Η Μέθοδος each**

Η μέθοδος each μας επιτρέπει να κάνουμε οτιδήποτε θέλουμε, σε κάθε αντικείμενο στο οποίο δείχνει (περιέχει δηλαδή) ο πίνακας. Έτσι αν θέλουμε να πούμε κάτι ωραίο σχετικά με κάθε γλώσσα στον παρακάτω πίνακα, θα γράφαμε τα εξής:

```
languages = ['English', 'German', 'Ruby']

languages.each do |lang|
  puts 'I love ' + lang + '!'
  puts 'Don\'t you?'
end

puts 'And let\'s hear it for C++!'
puts '...'
```

```
I love English!
Don't you?
I love German!
Don't you?
I love Ruby!
Don't you?
And let's hear it for C++!
...
```

Λοιπόν τι συνέβη; Είχαμε τη δυνατότητα να προσπελάσουμε όλα τα αντικείμενα του πίανκα, χωρίς να χρησιμοποιήσουμε αριθμούς ως δείκτες Αυτό είναι σίγουρα καλό. Μεταφράζοντας στη γλώσσα μας το προηγούμενο πρόγραμμα είναι σαν να λέμε: Για κάθε αντικείμενο (each) στον πίνακα languages, στρέψε τη μεταβλητή lang στο αντικείκενο και πράξε, οτιδήποτε θα σου πω, μέχρι να τελειώσεις.

Μπορεί να σκέφτεστε: "Αυτό μοιάζει πολύ με τους βρόχους που μάθαμε νωρίτερα" Ναι είναι παρόμοιο. Μια σημαντική διαφορά είναι πως η μέθοδος each είναι ακριβώς αυτό, μια μέθοδος δηλαδή. Αντίθετα οι while και end (όπως ακριβώς η do, η if, η else και όλες οι άλλες μπλε λέξεις) δεν είναι μέθοδοι. Είναι θεμελιώδη τμήματα της γλώσσας Ruby, όπως το σύμβολο = και οι

παρενθέσεις. Κάτι σαν τα σημεία στίξης στη δική μας γλώσσα ή οι δεσμευμένες λέξεις στην Pascal.

Όχι όμως και η each . Η each είναι ακόμα μία μέθοδος πινάκων. Οι μέθοδοι, όπως η each, οι οποίοι συμπεριφέροντας σαν βρόχοι, συχνά καλούνται επαναλήπτες . Ένα πράγμα που πρέπει να θυμάστε για τους επαναλήπτες είναι οτι πάντα ακολουθούνται από την λέξη do και τερματίζουν στην λέξη end. Η while και η if, ποτέ δεν είχαν do . Η εντολή do χρησιμοποιείται μόνο με επαναλήπτες.

Ακολουθεί ακόμα ένας μικρός, χαριτωμένος επαναλήπτης, αλλά δεν είναι μέθοδος πινάκων... είναι μέθοδος ακεραίων.

```
3.times do
  puts 'Hip-Hip-Hooray!'
end
```

```
Hip-Hip-Hooray!
Hip-Hip-Hooray!
Hip-Hip-Hooray!
```

**Περισσότερες μέθοδοι Πινάκων.**

Μέχρι εδώ μάθατε την each, αλλά υπάρχουνε πολλές άλλες μέθοδοι πινάκων... σχεδόν τόσες όσες και οι μέθοδοι αλφαριθμητικών! Στην πραγματικότητα μερικές από αυτές (όπως η length, η reverse, η +, και η *) δουλεύουν για τους πίνακες, με τον ίδιο τρόπο που δουλεύουν για τα αλφαριθμητικά. Βέβαια επιδρούν στις θέσεις του πίνακα και όχι στους χαρακτήρες των αλφαριθμητικών. Άλλες μέθοδοι, όπως η last και η join, χρησιμοποιούνται αποκλειστικά στους πίνακες. Ακόμα, άλλες όπως η push και η pop, στην πραγματικότητα, αλλάζουν τον πίνακα. Φυδικ, όπως και στις αλφαριθμητικές μεθόδους, δεν είναι απαραίτητο να τις θυμάστε όλες αυτές, αρκεί να θυμάστε, που θα τις βρείτε στο εγχειρίδιο σας.

Πρώτα ας κοιτάξουμε τις μεθόδους to_s και join. Η join δουλεύει αρκετά όπως και η to_s, εκτός από το οτι προσθέτει χαρακτήρες, ανάμεσα στα αντικείμενα των πινάκων. Ρίξτε μια ματιά:

```
foods = ['artichoke', 'brioche', 'caramel']

puts foods
puts
puts foods.to_s
puts
puts foods.join(', ')
puts
puts foods.join('  :)  ') + '  8)'

200.times do
  puts []
end
```

```
artichoke
brioche
caramel
```

```
artichokebriochecaramel

artichoke, brioche, caramel

artichoke  :)  brioche  :)  caramel  8)
```

Όπως μπορείτε να δείτε, η puts συμπεριφέρεται στους πίνακες διαφορετικά από τα άλλα αντικείμενα: Απλώς καλείται η puts, για κάθε ένα από τα αντικείμενα μέσα στον πίνακα. Γι' αυτό ακριβώς το λόγο καλώντας την puts 200 φορές για έναν άδειο πίνακα, δεν έχει κανένα αποτέλεσμα. Αυτό συμβαίνει γιατί ο πίνακας δεν δείχνει πουθενά (δεν περιέχει τίποτα), έτσι δεν υπάρχει κανένα περιεχόμενο για την puts (για να το εμφανίσει).Κάνοντας 200 φορές το τίποτα μας κάνει τίποτα. Δοκιμάστε να καλέσετε την puts, για έναν πίνακα που περιέχει άλλους πίνακες. Είναι το αποτέλεσμα αυτό που περιμένατε;

Επίσης προσέξατε οτι δεν χρησιμοποιήσα την puts με κενό αλφαριθμητικό όταν ήθελα να αφήσω κενή γραμμή, αλλά χρησιμοποίησα σκέτη την putsq Κάνουνε ακριβώς το ίδιο πράγμα.

Τώρα ας δούμε τις μεθόδους push, pop, και last. Οι μέθοδοι push και pop  είναι ένα είδος αντιθέτων, όπως η + και – είναι. Η push προσθέτει ένα αντικείμενο στο τέλος του πίνακα σας, και η pop, απομακρύνει το τελευταίο αντικείμενο από τον πίνακα (και σας λέει τι ήταν αυτό το τελευταίο αντικείμενο) .Η last είναι παρόμοια με την pop στο οτι σας λέει τι υπάρχει στο τέλος του πίνακα, εκτός από το οτι αφήνει τον πίνακα χωρίς αλλαγές. Έτσι η push και η pop πραγματικά αλλάζουν τον πίνακα:

```
favorites = []
favorites.push 'raindrops on roses'
favorites.push 'whiskey on kittens'

puts favorites[0]
puts favorites.last
puts favorites.lengthσ

puts favorites.pop
puts favorites
puts favorites.length
```

```
raindrops on roses
whiskey on kittens
2
whiskey on kittens
raindrops on roses
1
```

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

• Γράψτε το πρόγραμμα, το οποίο αναφέρθηκε στην αρχή αυτού του κεφαλαίου..
Σημείωση**:** Υπάρχει μια πολύ καλή μέθοδος πινάκων, η οποία σας δίνει μια ταξινομημένη εκδοχή ενός πίνακα: η sort. Χρησιμοποιήστε την.

• Δοκιμάστε να γράψετε το παραπάνω πρόγραμμα, χωρίς να χρησιμοποιήσετε τη μέθοδο sort. Η επίλυση προβλημάτων είναι πολύ σημαντικό κομμάτι στην εκμάθηση του προγραμματισμού. Δώστε στον εαυτό σας, λοιπόν, την ευκαιρία να εξασκηθεί όσο περισσότερο μπορείτε.

• Ξαναγράψτε το πρόγραμμα με τον Πίνακα Περιεχομένων (από το κεφάλαιο των μεθόδων). Ξεκινήστε το πρόγραμμα με έναν πίνακα, ο οποίος θα περιέχει, όλες τις πληροφορίες του Πίνακα Περιεχομένων (ονόματα κεφαλαίων, αριθμό σελίδων κλπ). Μετά τυπώστε στην οθόνη τις πληροφορίες, από τον πίνακα, μορφοποιημένες όμορφα όπως πρέπει σε έναν Πίνακα Περιεχομένων.

Μέχρι τώρα, έχετε μάθει ένα ικανοποιητικό πλήθος από διαφορετικές μεθόδους. Τώρα όμως είναι η ώρα να μάθετε να φτιάχνετε τις δικές σας μεθόδους.

8. Γράψτε τις δικές σας μεθόδους.

Όπως έχει ειπωθεί μέχρι τώρα, οι βρόχοι και και οι επαναλήπτες, σας επιτρέπουν να κάνετε το ίδιο πράγμα (να εκτελέσετε τον ίδιο κώδικα) ξανά και ξανά. Παρόλα αυτά, μερικές φορές, μπορεί να θέλετε να κάνετε το ίδιο πράγμα, αρκετές φορές, αλλά σε διαφορετικά σημεία στο πρόγραμμα. Για παράδειγμα,υποθέστε πως γράφετε ένα πρόγραμμα ερωτηματολογίου, για έναν φοιτητή ψυχολογίας. Μια πιθανή εκδοχή του ερωτηματολογίου θα μπορούσε να είναι και η εξής:

```
puts 'Hello, and thank you for taking the time to'
puts 'help me with this experiment.  My experiment'
puts 'has to do with the way people feel about'
puts 'Mexican food.  Just think about Mexican food'
puts 'and try to answer every question honestly,'
puts 'with either a "yes" or a "no".  My experiment'
puts 'has nothing to do with bed-wetting.'
puts

#  We ask these questions, but we ignore their answers.

goodAnswer = false
while (not goodAnswer)
  puts 'Do you like eating tacos?'
  answer = gets.chomp.downcase
  if (answer == 'yes' or answer == 'no')
    goodAnswer = true
  else
    puts 'Please answer "yes" or "no".'
  end
end

goodAnswer = false
while (not goodAnswer)
  puts 'Do you like eating burritos?'
  answer = gets.chomp.downcase
  if (answer == 'yes' or answer == 'no')
    goodAnswer = true
  else
    puts 'Please answer "yes" or "no".'
  end
```

```
end

#  We pay attention to *this* answer, though.
goodAnswer = false
while (not goodAnswer)
  puts 'Do you wet the bed?'
  answer = gets.chomp.downcase
  if (answer == 'yes' or answer == 'no')
    goodAnswer = true
    if answer == 'yes'
      wetsBed = true
    else
      wetsBed = false
    end
  else
    puts 'Please answer "yes" or "no".'
  end
end

goodAnswer = false
while (not goodAnswer)
  puts 'Do you like eating chimichangas?'
  answer = gets.chomp.downcase
  if (answer == 'yes' or answer == 'no')
    goodAnswer = true
  else
    puts 'Please answer "yes" or "no".'
  end
end

puts 'Just a few more questions...'

goodAnswer = false
while (not goodAnswer)
  puts 'Do you like eating sopapillas?'
  answer = gets.chomp.downcase
  if (answer == 'yes' or answer == 'no')
    goodAnswer = true
  else
    puts 'Please answer "yes" or "no".'
  end
end

#  Ask lots of other questions about Mexican food.

puts
puts 'DEBRIEFING:'
puts 'Thank you for taking the time to help with'
puts 'this experiment.  In fact, this experiment'
puts 'has nothing to do with Mexican food.  It is'
```

```
puts 'an experiment about bed-wetting.  The Mexican'
puts 'food was just there to catch you off guard'
puts 'in the hopes that you would answer more'
puts 'honestly.  Thanks again.'
puts
puts wetsBed
```

```
Hello, and thank you for taking the time to
help me with this experiment.  My experiment
has to do with the way people feel about
Mexican food.  Just think about Mexican food
and try to answer every question honestly,
with either a "yes" or a "no".  My experiment
has nothing to do with bed-wetting.

Do you like eating tacos?
yes
Do you like eating burritos?
yes
Do you wet the bed?
no way!
Please answer "yes" or "no".
Do you wet the bed?
NO
Do you like eating chimichangas?
yes
Just a few more questions...
Do you like eating sopapillas?
yes

DEBRIEFING:
Thank you for taking the time to help with
this experiment.  In fact, this experiment
has nothing to do with Mexican food.  It is
an experiment about bed-wetting.  The Mexican
food was just there to catch you off guard
in the hopes that you would answer more
honestly.  Thanks again.

false
```

Αυτό ήταν ένα πραγματικά μεγάλο πρόγραμμα, με πολλές επαναλήψεις εντολών (όλες οι ενότητες του κώδικα, που σχετίζονταν με ερωτήσεις για το μεξικάνικο φαγητό, ήταν πανομοιότυπες, ενώ αυτές για το “βρέξιμο” του κρεβατιού ήταν μόνο ελαφρώς διαφοροποιημένες ). Η επανάληψη παρόμοιων εντολών είναι κακό πράγμα (μεγαλώνει άσκοπα το μέγεθος του προγράμματος). Από την άλλη, πολλές φορές  δεν μπορούμε να το αποφύγουμε βάζοντας τον κώδικα αυτό σε έναν μεγάλο βρόχο ή επαναλήπτη, γιατί μερικές φορές θέλουμε μεταξύ των ερωτημάτων να παρεμβάλλουν διαφορετικά πράγματα. Σε καταστάσεις όπως αυτή, είναι προτιμότερο να γράψετε μία μέθοδο. Ορίστε πως:

```
def sayMoo
```

```
  puts 'mooooooo...'
end
```

Χμ.. το πρόγραμμα αυτό δεν εμφανίζει Moo στην οθόνη. Γιατί όχι; Επειδή δεν δώσατε τέτοια εντολή. Απλά είπατε στον υπολογιστή με ποιον τρόπο θα πει Moo, αλλά στην πραγματικότητα ποτέ δεν του είπατε να το πράξει. Κάντε και την παρακάτω προσπάθεια:

```
def sayMoo
  puts 'mooooooo...'
end

sayMoo
sayMoo
puts 'coin-coin'
sayMoo
sayMoo
```

```
mooooooo...
mooooooo...
coin-coin
mooooooo...
mooooooo...
```

Πολύ καλύτερα, δεν συμφωνείτε;

Με αυτό τον τρόπο ορίσατε τη μέθοδο sayMoo. (τα ονόματα των μεθόδων, όπως και των μεταβλητών, ξεκινάνε με ένα πεζό λατινικό γράμμα. Υπάρχουν, βέβαια ορισμένες μέθοδοι που εξαιρούνται όπως η + και η ==). Όπως ξέρετε όμως οι μέθοδοι, συσχετίζονται με αντικείμενα. Στην περίπτωση του προηγούμενου παραδείγματος, η μέθοδος που δημιουργήσατε, συσχετίζεται με το αντικείμενο που αναπαρίσταται από όλο το πρόγραμμα (Δηλαδή το mooooooo). Στο επόμενο κεφάλαιο, θα δείτε πως να προσθέτετε μεθόδους σε άλλα αντικείμενα. Πρώτα όμως...

**Παράμετροι Μεθόδων**

Πιθανά να έχετε προσέξει πως κάποιες μέθοδοι (όπως οι gets, to_s, reverse...), μπορείτε απλά να τις καλέσετε κατ' ευθείαν σε ένα αντικείμενο. Παρόλα αυτά, άλλες μέθοδοι (όπως οι +,-,puts...) "παίρνουνε" παραμέτρους για να πούνε στο αντικείμενο πως να υλοποιήσουνε τη μέθοδο. Για παράδειγμα, δεν θα μπορούσατε να πείτε 5+. Λέτε να προστεθεί το αντικείμενο 5, αλλά δεν λέτε με τι άλλο να προστεθεί το 5.

Για να προσθέσετε μια παράμετρο στη μέθοδο sayMoo (πχ τον αριθμό των moos), θα κάνατε αυτό:

```
def sayMoo numberOfMoos
  puts 'mooooooo...'*numberOfMoos
end
sayMoo 3
puts 'oink-oink'
sayMoo  #  This should give an error because the parameter is missing.
```

```
mooooooo...mooooooo...mooooooo...
oink-oink
```

```
#<ArgumentError: wrong number of arguments (0 for 1)>
```

Η numberOfMoos είναι μια μεταβλητή η οποία δείχνει στην παράμετρο που βάλατε, δηλαδή παίρνει την τιμή της από την παράμετρο. Έτσι αφού πληκτρολογήσατε sayMoo 3, τότε η παράμετρος είναι το 3, και η μεταβλητή numberOfMoos, δείχνει στο 3 (δηλαδή το 3 "αποθηκεύεται" στη μεταβλητή)

Όπως μπορείτε να δείτε, η παράμετρος τώρα ΑΠΑΙΤΕΙΤΑΙ. Άλλωστε με τι υποτίθεται πως η μέθοδος sayMoo θα πολλαπλασιάσει το αλφαριθμητικό 'mooooooo...', αν δεν δώσετε μια παράμετρο στη μέθοδο; Ο υπολογιστής δεν θα έχει ιδέα.

Αν τα αντικείμενα στη Ruby, είναι όπως τα ουσιαστικά και οι μέθοδοι είναι όπως τα ρήματα, τότε μπορείτε να έχετε στο μυαλό σας τα τις παραμέτρους ως επιρρήματα (όπως στη μέθοδο sayMoo, όπου η παράμετρος μας είπε πως -πόσες φορές καλύτερα- να εκτελεστεί η sayMoo) ή μερικές φορές ως άμεσα αντικείμενα (όπως στη μέθοδο puts, όπου η παράμετρος είναι αυτό που τυπώνει η puts, αυτό δηλαδή πάνω στο οποίο άμεσα ενεργεί )

**Τοπικές Μεταβλητές**

Στο επόμενο πρόγραμμα, υπάρχουν δύο μεταβλητές:

```
def doubleThis num
  numTimes2 = num*2
  puts num.to_s+' doubled is '+numTimes2.to_s
end

doubleThis 44
```

```
44 doubled is 88
```

Οι μεταβλητές είναι η num και η numTimes2 και οι δύο μεταβλητές, βρίσκονται μέσα στη μέθοδο doubleThis. Αυτές (και όλε ςοι μεταβλητές που έχουμε χρησιμοποιήσει μέχρι τώρα) είναι τοπικές μεταβλητές. Αυτό σημαίνει πως είναι "ζωντανές" (έχουνε ισχύ) μέσα στη μέθοδο και μόνο μέσα σε αυτή. Αν επιχειρήσετε να τις χρησιμοποιήσετε έξω από τις μεθόδους, θα πάρετε ως αποτέλεσμα ένα μήνυμα σφάλματος:

```
def doubleThis num
  numTimes2 = num*2
  puts num.to_s+' doubled is '+numTimes2.to_s
end

doubleThis 44
puts numTimes2.to_s
```

```
44 doubled is 88
#<NameError: undefined local variable or method `numTimes2' for #<StringIO:0x82ba21c>>
```

Undefined local variable... Δηλαδή απροσδιόριστη τοπική μεταβλητή. Στην πραγματικότητα έχετε προσδιορίσει αυτή την τοπική μεταβλητή, αλλά δεν είναι τοπική εκεί που προσπαθήσατε να την χρησιμοποιήσετε . Είναι τοπική μέσα στη μέθοδο.

Αυτό μπορεί να φαίνεται άβολο, αλλά πραγματικά δεν είναι έτσι, αντίθετα είναι πολύ καλό. Πρώτα από όλα σημαίνει οτι δεν έχετε πρόσβαση σε μεταβλητές, οι οποίες βρίσκονται μέσα στις

μεθόδους, αλλά και το αντίστροφο, πως αυτές δηλαδή δεν έχουν πρόσβαση στις άλλες μεταβλητές σας και έτσι δεν μπορείτε να τις μπερδέψετε:

```
def littlePest var
  var = nil
  puts 'HAHA!  I ruined your variable!'
end

var = 'You can\'t even touch my variable!'
littlePest var
puts var
```

```
HAHA!  I ruined your variable!
You can't even touch my variable!
```

Στο προηγούμενο μικρό πρόγραμμα, πόσες μεταβλητές πιστεύετε οτι υπάρχουν; Μία; η μεταβλητή var; Λάθος απάντηση. Στην πραγματικότητα υπάρχουν δύο μεταβλητές με το όνομα var: Μία μέσα στη μέθοδο littlePest και μία έξω από αυτή. Όταν καλέσαμε: littlePest Var στην πραγματικότητα μεταβιβάσαμε το αλφαριθμητικό από την μια var στην άλλη, έτσι ώστε και οι δύο δείχνανε (περιείχανε) το ίδιο αλφαριθμητικό. Έπειτα η μέθοδος littlePrest έδειξε την δική του τοπική μεταβλητή var προς την τιμή nil (δηλαδή το τίποτα), αλλά αυτό δεν επηρέασε καθόλου την μεταβλητή var έξω από τη μέθοδο.

**Επιστρεφόμενες τιμές**

Ίσως έχετε προσέξει, πως ορισμένες μέθοδοι σας επιστρέφουν "κάτι" πίσω όταν τις καλείτε. Για παράδειγμα η μέθοδος gets επιστρέφει ένα αλφαριθμητικό (το αλφαριθμητικό το οποίο πληκτρολογήσατε) και η μέθοδος +, παράδειγμα όταν χρησιμοποιείται: 5+3 επιστρέφει 8. Οι αριθμητικές μέθοδοι για αριθμούς επιστρέφουν αριθμούς και οι αριθμητικές μέθοδοι για αλφαριθμητικά επιστρέφουν αλφαριθμητικά.

Είναι σημαντικό να κατανοήσετε τη διαφορά ανάμεσα στις μεθόδους που επιστρέφουν κάποια τιμή μετά την κλήση τους και στην έξοδο πληροφοριών του προγράμματος στην οθόνη, όπως συμβαίνει με την κλήση της puts. Η επιστροφή δεν σημαίνει έξοδο στην οθόνη. Σημειώστε οτι η μέθοδος 5+3 επιστρέφει το 8, αλλά δεν το εμφανίζει στην οθόνη, δεν παράγει έξοδο δηλαδή.

Η puts με τη σειρά της τι επιστρέφει; Μέχρι τώρα δεν μας είχα απασχολήσει αυτό, αλλά ας το δούμε τώρα: So what *does* puts return? We never cared before, but let's look at it now:

```
returnVal = puts 'This puts returned:'
puts returnVal
```

```
This puts returned:
nil
```

Η πρώτη puts επέστρεψε nil (δηλ. Τίποτα). Αν και δεν το ελέγξαμε η δεύτερη puts έκανε το ίδιο. Η puts πάντα επιστρέφει nil. Κάθε μέθοδος πρέπει να επιστρέφει κάτι, ακόμα και αν αυτό είναι απλώς nil.So the first puts returned nil. Though we didn't test it, the second puts did, too; puts always returns nil. Every method has to return something, even if it's just nil.

Κάντε ένα διάλειμμα από το διάβασμα και γράψτε ένα πρόγραμμα για να βρείτε τι επέστρεψε η μέθοδος sayMoo .

Εκπλαγήκατε; Λοιπόν ορίστε πως δουλεύει: Η τιμή που επέστρεψε από μια μέθοδο είναι απλά η

τελευταία γραμμή της μεθόδου. Στην περίπτωση της sayMoo, αυτό σημαίνει πως επιστρέφει: puts 'mooooooo...'*numberOfMoos, η οποία είναι απλώς nil, από τη στιγμή που η puts επιστρέφει πάντα nil. Αν θέλαμε όλες οι μέθοδοι μας να επιστρέφουνε το αλφαριθμητικό 'yellow submarine', θα έπρεπε απλά να το γράψουμε στο τέλος αυτών:

```
def sayMoo numberOfMoos
  puts 'mooooooo...'*numberOfMoos
  'yellow submarine'
end

x = sayMoo 2
puts x
```

```
mooooooo...mooooooo...
yellow submarine
```

Ας δοκιμάσουμε, αυτό το ψυχολογικό πείραμα ξανά, αλλά αυτή τη φορά θα γράψουμε μία μέθεδο, η οποία θα θέτει τα ερωτήματα. Η μέθοδος θα χρειαστεί να παίρνει την κάθε ερώτηση ως παράμετρο και θα επιστρέφει true αν η απάντηση στην ερώτηση είναι Ναι και false αν η απάντηση είναι Όχι. Επίσης θα συντομέψουμε τον χαιρετισμό και τον απολογισμό, απλά για να είναι πιο εύκολο το πρόγραμμα μας:

```
def ask question
  goodAnswer = false
  while (not goodAnswer)
    puts question
    reply = gets.chomp.downcase

    if (reply == 'yes' or reply == 'no')
      goodAnswer = true
      if reply == 'yes'
        answer = true
      else
        answer = false
      end
    else
      puts 'Please answer "yes" or "no".'
    end
  end

  answer  #  This is what we return (true or false).
end

puts 'Hello, and thank you for...'
puts

ask 'Do you like eating tacos?'      #  We ignore this return value.
ask 'Do you like eating burritos?'
wetsBed = ask 'Do you wet the bed?'  #  We save this return value.
ask 'Do you like eating chimichangas?'
ask 'Do you like eating sopapillas?'
```

```
ask 'Do you like eating tamales?'
puts 'Just a few more questions...'
ask 'Do you like drinking horchata?'
ask 'Do you like eating flautas?'

puts
puts 'DEBRIEFING:'
puts 'Thank you for...'
puts
puts wetsBed
```

```
Hello, and thank you for...

Do you like eating tacos?
yes
Do you like eating burritos?
yes
Do you wet the bed?
no way!
Please answer "yes" or "no".
Do you wet the bed?
NO
Do you like eating chimichangas?
yes
Do you like eating sopapillas?
yes
Do you like eating tamales?
yes
Just a few more questions...
Do you like drinking horchata?
yes
Do you like eating flautas?
yes

DEBRIEFING:
Thank you for...

false
```

Όχι άσχημα, έτσι; Προσθέσαμε και περισσότερες ερωτήσεις και (και τώρα είναι εύκολο να προσθέτουμε ερωτήσεις), αλλά ακόμα και έτσι το πρόγραμμα μας είναι μικρότερο από το προηγούμενο. Αυτό είναι πρόοδος, δεν συμφωνείτε;

**Ένα μεγαλύτερο παράδειγμα:**

Νομίζω πως ακόμα ένα παράδειγμα μεθόδου θα ήταν χρήσιμο. Θα αποκαλούμε το επόμενο παράδειγμα (και τη μέθοδο) englishNumber. Η μέθοδος θα παίρνει έναν αριθμό, πχ το 22 και θα επιστρέφει την αλφαριθμητική του εκδοχή στα αγγλικά (δηλαδή σε αυτή την περίπτωση: twenty-two). Για αρχή, ας κάνουμε το παράδειγμα μας να δουλεύει για ακέραιους, από το 0 μέχρι το 100.

*(***NOTE:** *This method uses a new trick to return from a method early using the* return *keyword, and*

*introduces a new twist on branching:* elsif. *It should be clear in context how these work.)*

```
def englishNumber number
  #  We only want numbers from 0-100.
  if number < 0
    return 'Please enter a number zero or greater.'
  end
  if number > 100
    return 'Please enter a number 100 or lesser.'
  end

  numString = ''  #  This is the string we will return.

  #  "left" is how much of the number we still have left to write out.
  #  "write" is the part we are writing out right now.
  #  write and left... get it?  :)
  left  = number
  write = left/100         #  How many hundreds left to write out?
  left  = left - write*100  #  Subtract off those hundreds.

  if write > 0
    return 'one hundred'
  end

  write = left/10          #  How many tens left to write out?
  left  = left - write*10  #  Subtract off those tens.

  if write > 0
    if write == 1  #  Uh-oh...
      #  Since we can't write "tenty-two" instead of "twelve",
      #  we have to make a special exception for these.
      if    left == 0
        numString = numString + 'ten'
      elsif left == 1
        numString = numString + 'eleven'
      elsif left == 2
        numString = numString + 'twelve'
      elsif left == 3
        numString = numString + 'thirteen'
      elsif left == 4
        numString = numString + 'fourteen'
      elsif left == 5
        numString = numString + 'fifteen'
      elsif left == 6
        numString = numString + 'sixteen'
      elsif left == 7
        numString = numString + 'seventeen'
      elsif left == 8
        numString = numString + 'eighteen'
      elsif left == 9
        numString = numString + 'nineteen'
```

```
    end
    #  Since we took care of the digit in the ones place already,
    #  we have nothing left to write.
    left = 0
  elsif write == 2
    numString = numString + 'twenty'
  elsif write == 3
     numString = numString + 'thirty'puts 'Hello, and welcome to 7th grade English.'
puts    'My    name    is    Mrs.    Gabbard.     And    your    name    is...?'
name                                =                                gets.chomp

if                  name                    ==                  name.capitalize
      puts     'Please     take     a     seat,     '     +     name     +     '.'
else
    puts   name   +   '?    You   mean   '   +   name.capitalize   +   ',   right?'
     puts    'Don\'t    you    even    know    how    to    spell    your    name??'
                      reply                      =                    gets.chomp

               if              reply.downcase              ==              'yes'
                                                                            else
                                                                             end
end
```

Και πάλι, γράψτε το if, το else και το end την ίδια στιγμή. Βοηθάει να κρατάτε τα ίχνη του που βρίσκεστε κάθε στιγμή στον κώδικα. Να ξέρετε για ποιο if ισχύει το else και το end. Επίσης, βοηθάει στο να φαίνετια η δουλειά που έχετε να κάνετε, ευκολότερη, επειδή μπορείτε να επικεντρώσετε, σε ένα μικρό τμήμα, όπως να συμπληρώσετε τον κώδικα ανάμεσα στο if και το else. Ένα άλλο όφελος από αυτή την τακτική είναι πως ο υπολογιστής καταλαβαίνει το πρόγραμμα σε οποιοδήποτε στάδιο. Δηλαδή οποιαδήποτε από τις ανολοκλήρωτες εκδόσεις του προγράμματος, είδατε προηγουμένως θα μπορούσε να εκτελεστεί κανονικά . Δεν ήταν ολοκληρωμένα, αλλά ήταν λειτουργικά προγράμματα (δηλαδή προγράμματα που “δούλευαν”). Με αυτό τον τρόπο θα μπορούσατε να ελέγξετε το πρόγραμμα τη στιγμή που το γράφετε και να βλέπετε διαρκώς το παραγόμενο αποτέλεσμα, αλλά και που χρειάζεται ακόμα δουλειά να γίνει. Όταν περνάει ένα πρόγραμμα όλους αυτούς τους ελέγχους, τότε είστε σίγουρος πως αυτό που έπρεπε να κάνετε το κάνατε.

Αυτές οι συμβουλές, θα σας βοηθήσουν να γράψετε προγράμματα με διακλάδωση, αλλά επίεσης θα σας βοηθήσουν και με τον άλλο κύριο τύπο του ελέγχου ροής:

```
  elsif write == 4
    numString = numString + 'forty'
  elsif write == 5
    numString = numString + 'fifty'
  elsif write == 6
    numString = numString + 'sixty'
  elsif write == 7
    numString = numString + 'seventy'
  elsif write == 8
    numString = numString + 'eighty'
  elsif write == 9
    numString = numString + 'ninety'
  end

  if left > 0
    numString = numString + '-'
  end
 end
```

```
  write = left  #  How many ones left to write out?
  left  = 0     #  Subtract off those ones.

  if write > 0
    if    write == 1
      numString = numString + 'one'
    elsif write == 2
      numString = numString + 'two'
    elsif write == 3
      numString = numString + 'three'
    elsif write == 4
      numString = numString + 'four'
    elsif write == 5
      numString = numString + 'five'
    elsif write == 6
      numString = numString + 'six'
    elsif write == 7
      numString = numString + 'seven'
    elsif write == 8
      numString = numString + 'eight'
    elsif write == 9
      numString = numString + 'nine'
    end
  end

  if numString == ''
    #  The only way "numString" could be empty is if
    #  "number" is 0.
    return 'zero'
  end

  #  If we got this far, then we had a number somewhere
  #  in between 0 and 100, so we need to return "numString".
  numString
end

puts englishNumber(  0)
puts englishNumber(  9)
puts englishNumber( 10)
puts englishNumber( 11)
puts englishNumber( 17)
puts englishNumber( 32)
puts englishNumber( 88)
puts englishNumber( 99)
puts englishNumber(100)
```

```
zero
nine
ten
```

```
eleven
seventeen
thirty-two
eighty-eight
ninety-nine
one hundred
```

Βέβαια υπάρχουν αρκετά προβληματάκια σε αυτό το πρόγραμμα. Πρώτα από όλα, έχει πάρα πολλές επαναλήψεις. Δεύτερον, δεν μπορεί να χειριστεί αριθμούς, μεγαλύτερους από 100. Τρίτον, υπάρχουν πολλές ειδικές περιπτώσεις, πολλές επιστροφές κλπ. Μια καλή λύση θα ήταν η χρήση μερικών πινάκων για να "καθαρίσουμε" λιγάκι το πρόγραμμα:

```
def englishNumber number
  if number < 0  #  No negative numbers.
    return 'Please enter a number that isn\'t negative.'
  end
  if number == 0
    return 'zero'
  end

  #  No more special cases!  No more returns!

  numString = ''  #  This is the string we will return.

  onesPlace = ['one',     'two',      'three',    'four',     'five',
               'six',     'seven',    'eight',    'nine']
  tensPlace = ['ten',     'twenty',   'thirty',   'forty',    'fifty',
               'sixty',   'seventy',  'eighty',   'ninety']
   teenagers  =  ['eleven',   'twelve',      'thirteen', 'fourteen', 'fifteen','sixteen', 'seventeen', 'eighteen',
'nineteen']

  #  "left" is how much of the number we still have left to write out.
  #  "write" is the part we are writing out right now.
  #  write and left... get it?  :)
  left  = number
  write = left/100         #  How many hundreds left to write out?
  left  = left - write*100  #  Subtract off those hundreds.

  if write > 0
    #  Now here's a really sly trick:
    hundreds  = englishNumber write
    numString = numString + hundreds + ' hundred'
    #  That's called "recursion".  So what did I just do?
    #  I told this method to call itself, but with "write" instead of
    #  "number".  Remember that "write" is (at the moment) the number of
    #  hundreds we have to write out.  After we add "hundreds" to "numString",
    #  we add the string ' hundred' after it.  So, for example, if
    #  we originally called englishNumber with 1999 (so "number" = 1999),
    #  then at this point "write" would be 19, and "left" would be 99.
    #  The laziest thing to do at this point is to have englishNumber
    #  write out the 'nineteen' for us, then we write out ' hundred',
```

```
  #  and then the rest of englishNumber writes out 'ninety-nine'.

  if left > 0
    #  So we don't write 'two hundredfifty-one'...
    numString = numString + ' '
  end
 end

 write = left/10          #  How many tens left to write out?
 left  = left - write*10  #  Subtract off those tens.

 if write > 0
  if ((write == 1) and (left > 0))
    #  Since we can't write "tenty-two" instead of "twelve",
    #  we have to make a special exception for these.
    numString = numString + teenagers[left-1]
    #  The "-1" is because teenagers[3] is 'fourteen', not 'thirteen'.

    #  Since we took care of the digit in the ones place already,
    #  we have nothing left to write.
    left = 0
  else
    numString = numString + tensPlace[write-1]
    #  The "-1" is because tensPlace[3] is 'forty', not 'thirty'.
  end

  if left > 0
    #  So we don't write 'sixtyfour'...
    numString = numString + '-'
  end
 end

 write = left  #  How many ones left to write out?
 left  = 0     #  Subtract off those ones.

 if write > 0
  numString = numString + onesPlace[write-1]
  #  The "-1" is because onesPlace[3] is 'four', not 'three'.
 end

 #  Now we just return "numString"...
 numString
end

puts englishNumber(  0)
puts englishNumber(  9)
puts englishNumber( 10)
puts englishNumber( 11)
puts englishNumber( 17)
puts englishNumber( 32)
```

```
puts englishNumber( 88)
puts englishNumber( 99)
puts englishNumber(100)
puts englishNumber(101)
puts englishNumber(234)
puts englishNumber(3211)
puts englishNumber(999999)
puts englishNumber(1000000000000)
```

```
zero
nine
ten
eleven
seventeen
thirty-two
eighty-eight
ninety-nine
one hundred
one hundred one
two hundred thirty-four
thirty-two hundred eleven
ninety-nine hundred ninety-nine hundred ninety-nine
one hundred hundred hundred hundred hundred hundred
```

*Τώρα τα πράγματα είναι κάπως καλύτερα. Βέβαια το πρόγραμμα είναι αρκετά πυκνό, γι' αυτό και έχω γράψει τόσα πολλά σχόλια. Δουλεύει ακόμα και για μεγάλους αριθμούς, όχι βέβαια αρκετά καλά. Για παράδειγμα, σκέφτομαι πως "one trillion" θα ήταν καλύτερη επιστροφή για τον τελευταίο αριθμό, ή ακόμα και "one million million" (παρόλο που και οι τρεις επιστροφές είναι σωστές). Θα μπορούσατε να κάνετε εσείς αυτές τις αλλαγές τώρα...*

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

• Επεκτείνετε το παραπάνω πρόγραμμα englishNumber. Πρώτα από όλα, να περιέχει και τις χιλιάδες. Έτσι θα πρέπει να επιστρέφει 'one thousand' αντί για 'ten hundred' και 'ten thousand' αντί για 'one hundred hundred'.

• Επεκτείνετε το englishNumber λίγο ακόμα. Τώρα να περιέχει εκαομμύρια, έτσι ώστε να λαμβάνετε 'one million' αντί για 'one thousand thousand'. Μετά προσπαθήστε να προσθέσετε δισεκατομμύρια και τρισεκατομμύρια. Πόσο ψηλά μπορείτε να πάτε;

• Τώρα φτιάξτε το weddingNumber (πάντρεμα αριθμών) . Θα έπρεπε να δουλεύει σχεδόν όπως και το englishNumber, και επιπλέον θα εισάγει την λέξη"and" συνεχώς, έτσι ώστε να επιστρέφει τα νούμερα κάπως έτσι: 'nineteen hundred and seventy and two'.

• *"Ninety-nine bottles of beer..."* χρησιμοποιώντας το englishNumber και το παλιό πρόγραμμα που είχατε φτιάξει για το ποίημα, καταγράψτε τους στίχους του τραγουδιού σωστά αυτή τη φορά. Τιμωρήστε τον υπολογιστή σας, ξεκινήστε από τα 9999 μπουκάλια μπύρας. (Μην διαλέξετε κάποιο νούμερο εξαιρετικά μεγαλύτερο από αυτό, επειδή εμφάνιση όλων αυτών των επαναλήψεων στην οθόνη, θα χρειαστεί αρκετό χρόνο.

Συγχαρητήρια, τώρα είστε ένας αληθινός προγραμματιστής! Έχετε μάθει, όλα όσα χρειάζεστε για να χτίσετε τεράστια προγράμματα, από την αρχή. Αν έχετε ιδέες για προγράμματα που θα θέλατε να γράψετε για τον εαυτό σας, τολμήστε το!

Φυσικά χτίζοντας τα πάντα από την αρχή, μπορεί να είναι μια αρκετά αργή διαδικασία. Γιατί να ξοδεύετε χρόνο για να γράψετε κώδικα, τον οποίο έχει ήδη γράψει κάποιος άλλος; Θα θέλατε το πρόγραμμα σας να στέλενι ένα e-mail; Θα θέλατε να αποθηκεύετε και να φορτώνετε αρχεία στον υπολογιστή σας; Ή μήπως να παράγετε ιστοσελίδες για ένα εγχειρίδιο, όπου όλα τα παραδείγματα του κώδικα θα ελέγχονται αυτόματα; Η Ruby, έχει όλων των ειδών τα αντικείμενα, τα οποία μπορείτε να χρησιμοποιήσετε για να βοηθηθείτε να γράψετε πιο γρήγορα αποτελεσματικότερα προγράμματα.

## 9. Κλάσεις

Μέχρι τώρα έχουμε δει διαφορετικά είδη ή κλάσεις από αντικείμενα: strings, integers, floats, arrays και μερικά ειδικά αντικείμενα (true, false, και nil), για τα οποία θα μιλήσουμε αργότερα. Στην Ruby , αυτές οι κλάσεις πάντα γράφονται με το πρώτο γράμμα κεφαλαίο: String, Integer, Float, Array... κλπ. Γενικά, αν θέλετε να δημιουργήσετε έναν νέο αντικείμενο κάποιας συγκεκριμένης κλάσης χρησιμοποιείται την λέξη new:

```
a = Array.new  + [12345]  #  Array  addition.
b = String.new + 'hello'  #  String addition.
c = Time.new

puts 'a = '+a.to_s
puts 'b = '+b.to_s
puts 'c = '+c.to_s
```

```
a = 12345
b = hello
c = Tue Apr 14 16:29:20 GMT 2009
```

Επειδη, μπορούμε να δημιουργήσουμε arrays (πίνακες) και strings (αλφαριθμητικά) χρησιμοποιώντας [...] και '...' αντίστοιχα, πολύ σπάνια τα δημιουργούμε χρησιμοποιώντας το new. (Παρόλο που δεν είναι πραγματικά φανερό από το προηγούμενο παράδειγμα, το String.New δημιουργεί ένα άδειο string -αλφαριθμητικό- και το Array.New δημιουργεί έναν νέο array -πίνακα-). Επίσης οι κλάσεις, numbers είναι ειδικές εξαιρέσεις: δεν μπορείς να δημιουργήσεις έναν integer (ακέραιο), γράφοντας Integer.NEW. Απλά χρειάζεται να γράψεις τον ακέραιο.

### Η Κλάση Time

Τι συμβαίνει λοιπόν με την Κλάση Time; Τα αντικείμενα που ανήκουν στην Κλάση Time αντιπροσωπεύουν στιγμές στο χρόνο. Μπορείτε να προσθέσετε (ή να αφαιρέσετε) αριθμούς στις (ή από τις) ώρες, για να λάβετε νέες ώρες. Προσθέτοντας το 1.5 σε μια ώρα, δημιουργείτε μια νέα ώρα, ενάμιση δευτερόλεπτα αργότερα:

```
time  = Time.new   #  The moment I generated this web page.
time2 = time + 60  #  One minute later.

puts time
```

```
puts time2
```

```
Tue Apr 14 16:29:20 GMT 2009
Tue Apr 14 16:30:20 GMT 2009
```

Μπορείτε επίσης να δημιουργήσετε μια ώρα για μια συγκεκριμένη στιγμή χρησιμοποιώντας το Time.mktime:

```
puts Time.mktime(2000, 1, 1)        #  Y2K.
puts Time.mktime(1976, 8, 3, 10, 11)  #  When I was born.
```

```
Sat Jan 01 00:00:00 GMT 2000
Tue Aug 03 10:11:00 GMT 1976
```

Σημείωση: Η παρένθεση χρησιμεύει στην ομαδοποίηση των παραμέτρων για την mktime. Όο περισσότερες παραμέτρους προσθέτετε, τόσο περισσότερο ακριβής γίνεται η ώρα.. Μπορείτε να συγκρίνετε δύο ώρες, χρησιμοποιώντας τις μεθόδους σύγκρισης (μια πρωθύστερη ώρα είναι μικρότερη από μια μεταγενέσερη ώρα), και αν αφαιρέσετε μια ώρα από μία άλλη, θα λάβετε ως αποτέλεσμα των αριθμό των δευτερολέπτων που τις χρίζουν. Πειραματιστείτε με αυτές τις ιδιότητες.

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

• Ένα δισεκατομμύριο δευτερόλεπα... Ανακαλύψτε το ακριβές δευτερόλεπτο στο οποίο γεννηθήκατε (αν μπορείτε). Διαπιστώστε, πότε θα συμπληρώσετε ένα δισεκατομμύριο δευτερόλεπτα ζωής (ή αν είστε αρκετά μεγάλος, πότε τα συμπληρώσατε). Μετά σημειώστε το στα ημερολόγιο σας.

• Χρόνια Πολλά! Ρωτήστε κάποιον ποια χρονιά γεννήθηκε, στη συνέχεια τον μήνα, τέλος τη μέρα. Διαπιστώστε πόσο χρονών είναι και εμφανίστε ένα μεγάλο "ΧΡΟΝΙΑ ΠΟΛΛΑ" για κάθε γενέθλια που έχουν γιορτάσει στη ζωή τους.

**Η Κλάση Hash**

Μια ακόμα χρήσιμη κλάση είναι η κλάση Hash. Τα αντικείμενα της κλάσης Hash, μοιάζουν πολύ με πίνακες (arrays) : Έχουνε μια δέσμη από θύρες, οι οποίες μπορούν να δείχνουν σε διάφορα αντικείμενα (δηλαδή να περιέχουν διάφορα αντικείμενα). Παρόλα αυτά, σε έναν πίνακα, οι θυρίδες, παρτάσσονται σε μια σειρά και κάθε μία αριθμοδεικτείται (ξεκινώντας από το μηδέν). Σε ένα αντικείμενα hash, οι θυρίδες δεν τοποθετούνται στη σειρά, (είναι απλώς ένα συνοθύλευμα πραγμάτων, τοποθετημένα μαζί) και μπορείτε να χρησιμοποιήσετε οποιοδήποτε αντικείμενο για να αναφερθείτε σε μια θυρίδα, όχι απλώς έναν αριθμό-δείκτη. Είναι καλό να χρησιμποιείται αντικείμενα hashes, όταν έχετε μια μίξη, ένα συνοθύλευμα πραγμάτων, για τα οποία θέλετε να διατηρήσετε τα ίχνη τους, αλλά δεν ταριάζουν σε έναν ταξινομημένο κατάλογο . Για παράδειγμα τα χρώματα που χρησιμοποιώ για διαφορετικά κομμάτια κώδικα, με τον οποίο δημιουργήθηκε αυτό το εγχειρίδιο:

```
colorArray = []  #  same as Array.new
colorHash  = {}  #  same as Hash.new

colorArray[0]      = 'red'
colorArray[1]      = 'green'
colorArray[2]      = 'blue'
```

```
colorHash['strings']  = 'red'
colorHash['numbers']  = 'green'
colorHash['keywords'] = 'blue'

colorArray.each do |color|
  puts color
end
colorHash.each do |codeType, color|
  puts codeType + ':  ' + color
end
```

```
red
green
blue
strings:  red
keywords:  blue
numbers:  green
```

Αν χρησιμοποιήσετε πίνακα, θα πρέπει να θυμάστε οτι η θυρίδα 0 είναι για strings (αλφαριθμητικά), η θυρίδα 1 είναι για numbers (αριθμούς) κλπ. Αν όμως χρησιμοποιήσετε ένα αντικείμενο hash, είναι εύκολο. Η θυρίδα 'strings' κρατάει το χρώμα των strings, φυσικά. Δεν υπάρχει τίποτα που πρέπει να θυμάστε. Ίσως θα προσέξατε, πως όταν χρησιμοποιήσαμε το καθένα από αυτά τα αντικείμενα, στο hash, δεν εμφανίστηκαν με την ίδια σειρά που τα εισάγαμε στο hash. Οι πίνακες (arrays) είναι για να κρατάνε τα πράγματα σε μια σειρά και όχι τα hashes.

Παρόλο που οι περισσότεροι συνήθως χρησιμοποιούν τα strings (αλφαριθμητικά), για να δώσουν το όνομα μιας θυρίδας σε ένα hash, θα μπορούσατε να χρησιμοποιήσατε, οποιοδήποτε είδος αντικειμένου, ακόμα και πίνακες (arrays) ή και άλλα hashes (αν και δεν μπορώ να σκεφτώ έναν λόγο για να κάνετε κάτι τέτοιο):

```
weirdHash = Hash.new

weirdHash[12] = 'monkeys'
weirdHash[[]] = 'emptiness'
weirdHash[Time.new] = 'no time like the present'
```

Τα Hashes και οι πίνακες (arrays) είναι καλά και τα δύο , αλλά για διαφορετικά πράγματα το καθένα. Από εσάς εξαρτάται η απόφαση τι από τα δύο είναι καλύτερο για ένα συγκεκριμένο πρόβλημα.

**Επεκτείνοντας Κλάσεις**

Στο τέλος του προηγούμενου κεφαλαίου, γράψατε μια μέθοδο που επιστρέφει σε αλφαριθκητική μορφή (στα Αγγλικά) έναν δωσμένο ακέραιο αριθμό. Δεν ήταν μια μέθοδος ακεραίων ωστόσο. Ήταν μια γενικού προγράμματος μέθοδος . Δεν θα ήταν όμως ωραία, αν μπορούσατε να γράψετε κάτι σαν 22.to_eng, αντί να γράφετε englishNumber 22 ;Παρακάτω μπορείτε να δείτε με ποιον τρόπο είναι εφικτό κάτι τέτοιο:

```
class Integer

  def to_eng
    if self == 5
```

```
    english = 'five'
  else
    english = 'fifty-eight'
  end

  english
 end

end

#  I'd better test on a couple of numbers...
puts 5.to_eng
puts 58.to_eng
```

```
five
fifty-eight
```

Όπως βλέπετε, φαίνεται οτι δουλεύει, ικανοποιητικά.

Έτσι ορίσαμε μια μέθοδο ακεραίων (integer), εισερχόμενοι μέσα στην Κλάση Integer, προσδιορίζοντας τη μέθοδο εκεί μέσα, και στη συνέχεια εξερχόμαστε και πάλι από την Κλάση. Έτσι τώρα όλοι οι ακέραιοι, έχουν στη διάθεση τους αυτή τη μέθοδο, έστω και αν δεν είναι ολοκληρωμένη. Στην πραγματικότητα, αν δεν σας αρέσει ο τρόπος που μια προϋπάρχουσα μέθοδος όπως η to_s λειτουργεί, θα μπορούσατε να την ξανακαθορίσετε με τον ίδιο ακριβώς τρόπο, αλλά δεν σας το συνιστώ. Είναι προτιμότερο να αφήσετε τις παλιές μεθόδους έτσι ακριβώς όπως είναι και να δημιουργήσετε νέες, μόνο αν θέλετε να δημιουργήσετε μια πραγματικά νέα λειτουργία.

Ακόμα είστε μπερδεμένοι; επιτρέψτε μου να επανέλθω στο προηγούμενο πρόγραμμα και πάλι. Μέχρι τώρα, οποτεδήποτε εκτελέσατε κάποιον κώδικα ή καθορίσατε κάποια μέθοδο το κάνατε στο προεπιλεγμένο αντικείμενο πρόγραμμα. Στο τελευταίο πρόγραμμα, φύγατε από αυτό το αντικείμενο για πρώτη φορά και εισήλθατε στην Κλάση Integer. Καθορίσατε μια μέθοδο εκεί (άρα αυτή η μέθοδος είναι μέθοδος ακεραίων) και όλοι οι ακέραιοι μπορούν να την χρησιμοποιήσουν. Μέσα σε αυτή τη μέθοδο πλέον χρησιμοποιείται το self για να αναφερθείτε στο αντιμείμενο (την Integer δηλαδή) που χρησιμοποιεί αυτή τη μέθοδο.

**Δημιουργώντας Κλάσεις**

Μέχρι στιγμής έχετε δει έναν ικανοποιητικό αριθμό από διαφορετικές Κλάσεις και Αντικείμενα. Παρόλα αυτά είναι πιθανό να χρειαστείτε ήδη αντικειμένων, τα οποία η Ruby δεν διαθέτει. Ευτυχώς, το να δημιουργήσετε νέες Κλάσεις, είναι τόσο εύκολο όσο το να επεκτείνετε μια υπάρχουσα. Υποθέστε οτι θέλετε να ρίξετε μερικές ζαριές με την Ruby. Άρα θα πρέπει να δημιουργήσετε μια Κλάση Ζάρι. Ορίστε:

```
class Die

 def roll
  1 + rand(6)
 end

end

#  Let's make a couple of dice...
```

```
dice = [Die.new, Die.new]

#  ...and roll them.
dice.each do |die|
  puts die.roll
end
```

```
3
4
```

(Η εντολή, rand(6) επιστρέφει έναν τυχαίο αριθμό ανάμεσα στο 0 και το 5)

Είδατε πλέον έχετε ένα ολόδικό σας αντικείμενο.

Μπορείτε να ορίσετε όλων των ειδών τις μεθόδους για τα αντικείμενα σας... αλλά κάτι λείπει ακόμα. Με αυτά τα αντικείμενα, υπάρχει η αίσθηση, οτι προγραμματίζετε, χωρίς να έχετε μάθει για τις μεταβλητές. Κοιτάξτε στα Ζάρια σας για παράδειγμα. Μπορείτε να τα "ρίξετε" και κάθε φορά που θα το κάνετε θα σας δίνουν και έναν διαφορετικό αριθμό. Αλλά αν θέλατε να κρατηθείτε σε αυτόν τον αριθμό, θα έπρεπε να δημιουργήσετε μια μεταβλητή που να δείχνει στον αριθμό. Φαίνεται οτι κάθε σωστό ζάρι, θα έπρεπε να είναι σε θέση να έχει έναν αριθμό και κάθε φορά που ρίχνετε το ζάρι, θα έπρεπε να αλλάζει ο αριθμός. Αν παρακαλουθείτε το Ζάρι,  δεν θα έπρεπε να παρακολουθείτε και τον αριθμό που το Ζάρι δείχνει.

Όμως αν δοκιμάσετε να αποθηκεύσετε τον αριθμό που έδειξε το ζάρι σε μια (τοπική) μεταβλητή, ο αριθμός θα εξαφανιστεί όταν η ζαριά θα ολοκληρωθεί.  Χρειάζεστε να αποθηκεύσετε τον αριθμό σε ένα διαφορετικό είδος μεταβλητής:

**Στιγμιαίες Μεταβλητές ή Μεταβλητές Στιγμιότυπο**

Φυσιολογικά, όταν θέλετε να μιλήσετε για ένα αλφαριθμητικό θα το αποκαλείται απλώς αλφαριθμητικό. Επίσης θα μπορούσατε να το αποκαλείτε ένα αλφαριθμητικό αντικείμενο.. Μερικές φορές οι προγραμματιστές μπορεί να το αποκαλούν, ένα στιγμιότυπο/υπόδειγμα της Κλάσης Αλφαριθμητικό, αλλά αυτό είναι απλά ένας φανταχτερός τρόπος (και μάλλον μακροσκελής) για να πούμε απλά αλφαριθμητικό. Ένα στιγμιότυπο της Κλάσσης είναι απλά ένα αντικείμενο αυτής της Κλάσης.

Άρα τα στιγμιότυπα μεταβλητών είναι απλά μεταβλητές ενός αντικειμένου. Οι τοπικές μεταβλητές μιας μεθόδου διαρκούν μέχρι η να τερματιστεί η μέθοδος. Οι στιγμιαίες μεταβλητές ή στιγμιότυπα μεταβλητών, από την άλλη διαρκούν όσο και το αντικείμενο.  Για να είναι διακριτές οι στιγμιαίες μεταβλητές από τις τοπικές μεταβλητές, θα χρησιμοποιείται το σύμβολο @ μπροστά από τα ονόματα τους.

```
class Die

  def roll
    @numberShowing = 1 + rand(6)
  end

  def showing
    @numberShowing
  end

end
```

```
die = Die.new
die.roll
puts die.showing
puts die.showing
die.roll
puts die.showing
puts die.showing
```

```
1
1
5
5
```

Η μέθοδος roll ρίχνει το ζάρι και η showing μας λέει ποιο νούμερο δίχνει το ζάρι. Όμως τι θα γίνει αν προσπαθήσετε να δείτε το αποτέλεσμα της showing, πριν όμως "ρίξετε" το ζάρι; (δηλαδή πριν θέσετε τη @numberShowing)?

```
class Die

  def roll
    @numberShowing = 1 + rand(6)
  end

  def showing
    @numberShowing
  end

end

#  Since I'm not going to use this die again,
#  I don't need to save it in a variable.
puts Die.new.showing
```

```
nil
```

Τουλάχιστον δεν μας επέστρεψε μήνυμα λάθους. Βέβαια δεν έχει νόημα για ένα ζάρι η τιμή nil και οτιδήποτε αυτή μπορεί να σημαίνει εδώ. Θα ήταν πολύ καλό αν μπορούσαμε να "εγκαθιδρύσουμε¨το νέο μας αντικείμενο ζάρι, ακριβώς τη στιγμή που αυτό δημιουργήθηκε. Γι' αυτό υπάρχει η μέθοδος initialize:

```
class Die

  def initialize
    #  I'll just roll the die, though we
    #  could do something else if we wanted
    #  to, like setting the die with 6 showing.
    roll
  end

  def roll
```

```
  @numberShowing = 1 + rand(6)
 end

 def showing
  @numberShowing
 end

end

puts Die.new.showing
```

6

Όταν ένα αντικείμενο δημιουργείται, η αντίστοιχη του μέθοδος initialize (αν έχει καθοριστεί μία) καλείται.

Τα ζάρια μας είναι σχετικά τέλεια. Το μόνο πράγμα που πιθανά λείπει είναι ένας τρόπος, να ορίζεται ποια πλευρά του ζαριού φαίνεται, ποιο θα είναι το αποτέλεσμα δηλαδή της ζαριάς.. Γιατί δεν γράφετε μια μέθοδο, η οποία θα το κάνει αυτό;

Συνεχίστε το εγχειρίδιο, όταν έχετε τελειώσει (και φυσικά να ελέγξετε οτι δούλεψε η μέθοδος). Σιγουρέψτε οτι κανείς δεν θα μπορεί να θέσει το ζάρι να έχει ως αποτέλεσμα το 7.

Ας δούμε ακόμα ένα περισσότερο ενδιαφέρον παράδειγμα. Υποθέστε οτι θέλετε να δημιουργήσετε ένα εικονικό κατοικίδιο, ένα μωρό δράκο. Όπως τα περισσότερο μωρά, θα έπρεπε να μπορεί να τρώει, να κοιμάται και να poop, το οποίο σημαίνει οτι θα χρειαστεί να είστε ικανοί να το ταΐσετε, να το βάζετε στο κρεβάτι, και να το πηγαίνετε βόλτες. Εσωτερικά, ο δράκος σας θα χρειάζεται να καταλαβαίνει πότε πεινάει, πότε είναι κουρασμένος και πότε χρειάζεται να πάει βόλτα, αλλά εσείς δεν α μπορείτε να τα καταλάβετε αυτά, όταν έρχεστε σε επαφή με το δράκο, όπως δεν μπορείτε να ρωτήσετε ένα μωρό ανθρώπων: “Μήπως πεινάς;” Μπορείτε επίσης να προσθέσετε, μερικούς άλλους αστείους τρόπους για να αλληλεπιδράσετε με το μωρό-δράκος και μόλις γεννηθεί θα του δώσετε και ένα όνομα (Οτιδήποτε μεταβιβάσετε στη νέα μέθοδο, μεταβιβάζεται και στη μέθοδο initialize για σας). Δείτε το στην πράξη:

Our dice are just about perfect. The only thing that might be missing is a way to set which side of a die is showing... why don't you write a cheat method which does just that! Come back when you're done (and when you tested that it worked, of course). Make sure that someone can't set the die to have a 7 showing!

So that's some pretty cool stuff we just covered. It's tricky, though, so let me give another, more interesting example. Let's say we want to make a simple virtual pet, a baby dragon. Like most babies, it should be able to eat, sleep, and poop, which means we will need to be able to feed it, put it to bed, and take it on walks. Internally, our dragon will need to keep track of if it is hungry, tired, or needs to go, but we won't be able to see that when we interact with our dragon, just like you can't ask a human baby, "Are you hungry?". We'll also add a few other fun ways we can interact with our baby dragon, and when he is born we'll give him a name. (Whatever you pass into the new method is passed into the initialize method for you.) Alright, let's give it a shot:

```
class Dragon

 def initialize name
  @name = name
  @asleep = false
  @stuffInBelly     = 10  #  He's full.
  @stuffInIntestine =  0  #  He doesn't need to go.
```

```
  puts @name + ' is born.'
end

def feed
  puts 'You feed ' + @name + '.'
  @stuffInBelly = 10
  passageOfTime
end

def walk
  puts 'You walk ' + @name + '.'
  @stuffInIntestine = 0
  passageOfTime
end

def putToBed
  puts 'You put ' + @name + ' to bed.'
  @asleep = true
  3.times do
    if @asleep
      passageOfTime
    end
    if @asleep
      puts @name + ' snores, filling the room with smoke.'
    end
  end
  if @asleep
    @asleep = false
    puts @name + ' wakes up slowly.'
  end
end

def toss
  puts 'You toss ' + @name + ' up into the air.'
  puts 'He giggles, which singes your eyebrows.'
  passageOfTime
end

def rock
  puts 'You rock ' + @name + ' gently.'
  @asleep = true
  puts 'He briefly dozes off...'
  passageOfTime
  if @asleep
    @asleep = false
    puts '...but wakes when you stop.'
  end
end
```

```
private

#  "private" means that the methods defined here are
#  methods internal to the object.  (You can feed
#  your dragon, but you can't ask him if he's hungry.)

def hungry?
  #  Method names can end with "?".
  #  Usually, we only do this if the method
  #  returns true or false, like this:
  @stuffInBelly <= 2
end

def poopy?
  @stuffInIntestine >= 8
end

def passageOfTime
  if @stuffInBelly > 0
    #  Move food from belly to intestine.
    @stuffInBelly     = @stuffInBelly     - 1
    @stuffInIntestine = @stuffInIntestine + 1
  else  #  Our dragon is starving!
    if @asleep
      @asleep = false
      puts 'He wakes up suddenly!'
    end
    puts @name + ' is starving!  In desperation, he ate YOU!'
    exit  #  This quits the program.
  end

  if @stuffInIntestine >= 10
    @stuffInIntestine = 0
    puts 'Whoops!  ' + @name + ' had an accident...'
  end

  if hungry?
    if @asleep
      @asleep = false
      puts 'He wakes up suddenly!'
    end
    puts @name + '\'s stomach grumbles...'
  end

  if poopy?
    if @asleep
      @asleep = false
      puts 'He wakes up suddenly!'
    end
    puts @name + ' does the potty dance...'
```

```
    end
  end

end

pet = Dragon.new 'Norbert'
pet.feed
pet.toss
pet.walk
pet.putToBed
pet.rock
pet.putToBed
pet.putToBed
pet.putToBed
pet.putToBed
```

```
Norbert is born.
You feed Norbert.
You toss Norbert up into the air.
He giggles, which singes your eyebrows.
You walk Norbert.
You put Norbert to bed.
Norbert snores, filling the room with smoke.
Norbert snores, filling the room with smoke.
Norbert snores, filling the room with smoke.
Norbert wakes up slowly.
You rock Norbert gently.
He briefly dozes off...
...but wakes when you stop.
You put Norbert to bed.
He wakes up suddenly!
Norbert's stomach grumbles...
You put Norbert to bed.
He wakes up suddenly!
Norbert's stomach grumbles...
You put Norbert to bed.
He wakes up suddenly!
Norbert's stomach grumbles...
Norbert does the potty dance...
You put Norbert to bed.
He wakes up suddenly!
Norbert is starving!  In desperation, he ate YOU!
```

*Φυσικά, θα ήταν καλύτερα αν αυτό το πρόγραμμα ήταν αλληλεπιδραστικό (ή διαλογικό), αλλά μπορείτε να υλοποιήσετε αυτό το στοιχεία, αργότερα. Απλά προσπάθησ ανα σας δείξω, εκείνα τα τμήματα που σχετίζονται άμεσα με τη δημιουργία μιας νέας κλάσης δράκου.*

*Είδατε μερικά νέα πράγματα σε αυτό το παράδειγμα. Το πρώτο από αυτά είναι πολύ απλό: Η εντολή exit τερματίζει το πρόγραμμα σε εκείνο το σημείο.  Το δεύτερο είναι πως η λέξη private, η οποία τοποθετήθηκε στη μέση του καθορισμού της κλάσης μας. Θα μπορούσα να μην την είχα χρησιμοποιήσει, αλλά ήθελα να ενδυναμώσω την ιδέα οτι κάποιες μέθοδοι είναι απλά πράγματα που*

*μπορείτε να κάνετε σε έναν δράκο (κλάση δράκο) και άλλα, τα οποία απλά συμβαίνουν μέσα σε έναν δράκο. Μπορείτε να τα σκέφτεστε αυτά τα πράγματα "μέσα στην κουκούλα" : αν δεν είστε μηχανικός αυτοκινήτων, το μόνο που πραγματικά χρειάζεται να γνωρίζετε είναι το πετάλι του γκαζιου το πεντάλ του φρένου και το τιμόνι. Ένας προγραμματιστής θα αποκαλούσε αυτά τα πράγματα ως τη δημόσια διεπαφή με το αμάξι σας. Πως όμως ο αερόσακκος σας γνωρίζει πότε θα ανοίξει; αυτή η διαδικασία είναι εσωτερική στο αυτοκίνητο σας, ένας απλός οδηγός δεν χρειάζεται να γνωρίζει σχετικά με αυτό το θέμα.*

Ας δούμε ένα πιο συγκεκριμένο παράδειγμα σχετικό με αυτά που μόλις είπαμε, ας μιλήσουμε πως θα αναπαριστούσατε ένα αυτοκίνητο σε ένα βιντεοπαιχνίδι. Πρώτα από όλα θα έπρεπε να αποφασίσετε, πως θα θέλατε να μοιάζει η δημόσια διεπαφή σας. Με άλλα λόγια, ποιες μεθόδους θα έπρεπε οι άνθρωποι να είναι ικανοί να καλέσουν σε ένα από τα αντικείμενα αυτοκίνητα σας; Λοιπόν, αυτοί χρειάζεται να είναι ικανοί να πιέσουν το πετάλι του γκαζιού και το πετάλι του φρένου, αλλά θα χρειάζονταν επίσης να καθορίζουν σε τι βαθμό θα πιέζουν το πετάλι (υπάρχει πολύ μεγάλη διαφορά ανάμεσα στο χάιδεμα και το σανίδωμα του γκαζιού). Θα χρειάζονταν επίσης να είναι ικανοί να στρίψουν και επίσης θα χρειάζονταν να είναι κανοί να πούνε ποσό πολύ θα στρίψουν τους τροχούς; Υποθέτω οτι θα μπορούσατε να προχωρήσετε ακόμα παραπέρα και να προσθέσετε έναν συμπλέκτη, φλάσ, εκτοξευτή ρουκετών κλπ. Εξαρτάται από τον τύπο του παιχνιδιού που φτιάχνετε.

Εσωτερικά σε ένα αντικείμενο αυτοκινήτου, ωστόσο, θα χρειάζονταν να υπάρχουν πολλά περισσότερα: ένα αυτοκίνητο θα χρειαζόταν μια ταχύτητα, μια κατεύθυνση και μια θέση (αυτά είναι μόνο τα πιο βασικά). Αυτά τα χαρακτηριστικά θα τροποποιούνταν από το πάτημα των πεταλιών του γκαζιού ή του φρένου και του στριψίματος του τιμονιού, φυσικά, αλλά ο χρήστης δεν θα μπορούσε να θέσει τις τιμές τους απευθείας (κάτι τέτοιο θα διαστρέβλωνε την πραγματικότητα). Πιθανά επίσης, θα θέλετε να παρακολουθείτε, οποιαδήποτε βλάβη ή αστοχία στο αυτοκίνητο σας. Αυτές οι λειτουργίες θα έπρεπε να είναι όλες εσωτερικά στο αντικείμενο σας, αυτοκίνητο.

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

• Δημιουργήστε μια κλάση Πορτοκαλόδεντρο (OrangeTree). Αυτή θα πρέπει να έχει μια μέθοδο ΎψοςMake (height), η οποία θα επιστρέφει το ύψος του προτοκαλόδεντρου και μια μέθοδο ΈναςΧρόνοςΠερνάει , η οποία όταν καλείται "γερνάει" το δέντρο κατά ένα χρόνο. Κάθε χρόνο το δέντρο ψηλώνει (όσο νομίζετε οτι ένα δέντρο πορτοκαλιάς θα έπρεπε να αναπτυχθεί σε ένα χρόνο) και ύστερα από ορισμένο αριθμό ετών (ξανά μετά από δική σας κλήση) το δέντρο θα έπρεπε να πεθάνει. Για τα πρώτα (λίγα) χρόνια το δέντρο δεν θα πρέπει να παράγει φρούτα, αλλά στη συνέχεια θα πρέπει να παράγει και υποθέτω πως η παραγωγή εξαρτάται από την ηλικία του δέντρου. Καθορίστε εσείς την παραγωγή, με βάση οτι σας φαίνεται πιο λογικό. Φυσικά θα πρέπει να είστε σε θέση α)να μετράτε τα πορτοκάλια με την μέθοδο countTheOranges (η οποία επιστρέφει τον αριθμό των πορτοκαλιών στο δέντρο), να μαζεύετε ένα πορτοκάλι με τη μέθοδο pickAnOrange (η οποία μειώνει τον μετρητή πορτοκαλιών κατά ένα και επιστρέφει ένα αλφαριθμητικό, το οποίο σας ενημερώνει πόσο απολαυστικό ήταν το πορτοκάλι ή αλλιώς σας λέει οτι δεν υπάρχουν άλλα πορτοκάλια να μαζέψετε αυτό τον χρόνο, αν έχουν τελειώσει). Σιγουρευτείτε, οτι τα πορτοκάλια που δεν συλλέξατε σε έναν χρόνο, πέφτουν από το δέντρο πριν τον ερχομό του επόμενου χρόνου.

• Γράψτε ένα πρόγραμμα, έτσι ώστε να μπορείτε να αλληλεπιδράσετε με το μωρό δράκο που έχετε δημιοργήσει. Το πρόγραμμα αυτό θα πρέπει να σας δίνει τη δυνατότητα να δώσετε εντολές όπως τροφή και βόλτα και να καλούνται οι αντίστοιχες μέθοδοι στο δράκο σας. Φυσικά από τη στιγμή που εσείς θα εισάγετε, απλω αλφαριθμητικά, θα πρέπει να έχετε έναν τρόπο αποστολής της

κατάλληλης μεθόδου, όπου το πρόγραμμα σας ελέγχει πιο αλφαριθμητικό εισήχθηκε και στη συνέχεια καλεί την κατάλληλη μέθοδο.

Και αυτό είναι όλο όσο έχετε να κάνετε. Μέχρι στιγμής δεν σας έχουμε πει τίποτα για καμία από αυτές τις κλάσεις για να κάνετε πράγματα όπως να στέλνετε email, να αποθηκεύετε ή να φορτώνετε αρχεία στον υπολογιστή σας ή πως να δημιουργείτε παράθυρα και κουμπιά εντολών ή τρισδιάστατους κόσμους ή πάρα πολλά άλλα πράγματα. Λοιπόν υπάρχουν τόσες πολλές κλάσεις που μπορείτε να χρησιμοποιήσετε, ώστε είναι αδύνατο να παρουσιαστούν όλες. Ωστόσο μπορείτε να ασχοληθείτε μόνοι σας με αυτές τις οποίες θα χρειαστείτε και θα σας πούμε, που πρέπει να απευθυνθείτε για να βρείτε αυτές τις πληροφορίες. Πρώτα όμως υπάρχει ακόμα ένα χαρακτηριστικό της Ruby, το οποία θα έπρεπε να γνωρίζετε, ένα χαρακτηριστικό το οποίο οι περισσότερες γλώσσες δεν το έχουν, αλλά κάτι με το οποίο πολύ απλά δεν θα μπορούσατε να προγραμματίσετε ικανοποιητικά:

## 10. Blocks and Procs

Είναι σίγουρα ένα από τα θετικότερα χαρακτηριστικά της γλώσσας Ruby. Κι άλλες γλώσσες προγραμματισμού έχουν αυτό το χαρακτηριστικό, αν και μπορεί να το αποκαλούν διαφορετικά (όπως εγκλεισμοί), αλλά οι περισσότερες από τις πιο δημοφιλείς δεν έχουν αυτό το χαρακτηριστικό και κάτι τέτοιο είναι ντροπή.

Τι είναι λοιπόν αυτό το τόσο καλό νέο πράγμα; Είναι η ικανότητα να πάρετε μια ομάδα κώδικα (κώδικας ανάμεσα σε ένα do και ένα end), να την "τυλίξετε" μέσα σε ένα αντικείμενο , να την αποθηκεύσετε μέσα σε μια μεταβλητή ή να την μεταβιβάσετε σε μια μέθοδο και να εκτελέσετε τον κώδικα μέσα στο block, όποτε θέλετε (περισσότερες από μία φορές αν θέλετε). Μοιάζει δηλαδή με ένα είδος μεθόδου, εκτός από το οτι δεν προορίζεται για κάποιο αντικείμενο (είναι ένα αντικείμενο) και μπορείτε να το αποθηκεύσετε ή να το μεταβιβάσετε, όπως ακριβώς μπορείτε να κάνετε με οποιοδήποτε αντικείμενο. Δείτε όμως ένα παράδειγμα:

```
toast = Proc.new do
  puts 'Cheers!'
end

toast.call
toast.call
toast.call
```

```
Cheers!
Cheers!
Cheers!
```

Έτσι δημιούργησα ένα proc (το οποίο νομίζω πως είναι συντόμευση της λέξης "procedure", δηλαδή διαδικασία), το οποίο περιλαμβάνει το block του κώδικα και στη συνέχεια κάλεσα το proc τρεις φορές. Όπως μπορείτε να δείτε μοιάζει πάρα πολύ με μέθοδο

Πραγματικά, μοιάζει ακόμα περισσότερο σαν μέθοδο, από οτι ήδη σας έδειξα με το παράδειγμα, γιατί τα blocks μπορούν να πάρουν και παραμέτρους:.

```
doYouLike = Proc.new do |aGoodThing|
```

```
  puts 'I *really* like '+aGoodThing+'!'
end

doYouLike.call 'chocolate'
doYouLike.call 'ruby'
```

```
I *really* like chocolate!
I *really* like ruby!
```

Έτσι λοιπόν είδαμε με παραδείγματα τι είναι τα blocks και τα procs και φυσικά πως χρησιμοποιούνται. Αλλά ποιο είναι το νόημα ύπαρξης τους; Γιατί δεν χρησιμοποιούμε απλά τις μεθόδους; Απλά γιατί υπάρχουν ορισμένα πράγματα που δεν μπορείτε να κάνετε με τις μεθόδους. Συγκεκριμένα δεν μπορείτε να μεταβιβάσετε τις μεθόδους μέσα σε άλλες μεθόδους (αλλά μπορείτε να μεταβιβάσετε τα procs μέσα σε μεθόδους) και οι μέθοδοι δεν μπορούν να επιστρέψουν άλλες μεθόδους ως αποτέλεσμα (αλλά μπορούν να επιστρέψουν procs) . Αυτό συμβαίνει, γιατί απλά τα procs είναι αντικείμενα, ενώ οι μέθοδοι δεν είναι.

Όλα αυτά σας μοιάζουν οικεία; Είναι γιατί είδατε τα blocks και νωρίτερα, όταν μάθατε για τους επαναλήπτες. Ας μιλήσουμε όμως γι' αυτά λίγο ακόμα:

**Μέθοδοι, που δέχονται Procs**

Όταν μεταβιβάζουμε ένα proc σε μια μέθοδο μπορούμε να ελέγξουμε πως, αν και πόσες φορές καλούμε το proc. Για παράδειγμα, ας πούμε οτι υπάρχει κάτι που θέλουμε να κάνουμε πριν και αφού τρέξουμε ένα τμήμα του κώδικα:

```
def doSelfImportantly someProc
  puts 'Everybody just HOLD ON!  I have something to do...'
  someProc.call
  puts 'Ok everyone, I\'m done.  Go on with what you were doing.'
end

sayHello = Proc.new do
  puts 'hello'
end

sayGoodbye = Proc.new do
  puts 'goodbye'
end

doSelfImportantly sayHello
doSelfImportantly sayGoodbye
```

```
Everybody just HOLD ON!  I have something to do...
hello
Ok everyone, I'm done.  Go on with what you were doing.
Everybody just HOLD ON!  I have something to do...
goodbye
Ok everyone, I'm done.  Go on with what you were doing.
```

Ίσως το παραπάνω παράδειγμα να μην σας γεμίζει το μάτι, όμως είναι πολύ ικανοποιητικό παράδειγμα. Είναι πολύ συνηθισμένο, στον προγραμματισμό να τίθενται αυστηρές προδιαγραφές για το τι πρέπει να γίνει και πότε. Αν θέλετε να αποθηκεύσετε ένα αρχείο για παράδειγμα, πρέπει πρώτα να "ανοίξετε" το αρχείο, να γράψετε τις πληροφορίες που θέλετε στο αρχείο και στο τέλος να "κλείσετε" το αρχείο. Αν ξεχάσετε να κλείσετε το αρχείο, μπορεί να υπάρξουν προβλήματα με το αρχείο σας (πχ απώλεια δεδομένων). Έτσι κάθε φορά που θέλετε α αποθηκεύσετε ή να φορτώσετε ένα αρχείο, πρέπει να ακολουθήσετε την ίδια διαδικασία: "ανοίγετε" το αρχείο, κάνετε την επεξεργασία που θέλετε να κάνετε και στη συνέχεια "κλείνετε" το αρχείο. Αυτή η διαιδκασία είναι κουραστική και είναι πολύ εύκολο να ξεχάσετε κάποιο στάδιο. Στην Ruby όμως, η αποθήκευση (ή η μεταφόρτωση) δουλεύει παρόμοια με τον παραπάνω κώδικα, έτσι δεν πρέπει να ανησυχείτε εσείς για οτιδήποτε, παρά μόνο γι' αυτό που πραγματικά θέλετε να αποθηκεύσετε ή να μεταφορτώσετε. (Στο επόμενο κεφάλαιο θα μάθετε πως να κάνετε εργασίες όπως η αποθήκευση και η μεταφόρτωση).

Επίσης μπορείτε να γράψετε μεθόδους, οι οποίες θα καθορίζουν πόσες φορές, ή ακόμα και αν θα καλείτε ένα proc. Ακολουθεί μια μέθοδος, η οποία θα καλεί ένα proc και θα το εκτελεί κατά το ήμισι του χρόνου, και μι αάλλη μέθοδος που θα το καλεί δύο φορές:

```
def maybeDo someProc
  if rand(2) == 0
    someProc.call
  end
end

def twiceDo someProc
  someProc.call
  someProc.call
end

wink = Proc.new do
  puts '<wink>'
end

glance = Proc.new do
  puts '<glance>'
end

maybeDo wink
maybeDo glance
twiceDo wink
twiceDo glance
```

```
<glance>
<wink>
<wink>
<glance>
<glance>
```

Αυτές είναι ορισμένες από τις πιο κοινές χρήσεις των procs, οι οποίες μας επιτρέπουν να κάνουμε πράγματα που πολύ απλά δεν θα μπορούσαμε να κάνουμε χρησιμοποιώντας μόνο μεθόδους. Σίγουρα θα μπορούσατε να γράψετε μια μέθοδο που θα εμφανίζει δυο φορές τη λέξη

wink, αλλά δεν θα μπορούσατε να γράψετε μία μέθοδο που θα έκανε ***κάτι*** δύο φορές.

Πριν προχωρήσετε ρίξτε μια ματιά σε ένα τελευταίο παράδειγμα. Μέχρι τώρα τα procs, τα οποία έχουμε χρησιμοποιήσει στα παραδείγματα, μοιάζουν αρκετά μεταξύ τους, είναι παρόμοια δηλαδή. Αυτή τη φορά, τα procs, θα είναι αρκετά διαφορετικά, έτσι ώστε να μπορέσετε να δείτε πως μια μέθοδος εξαρτάται από το procs, το οποίο μεταβιβάζεται σε αυτή. Η μέθοδος μας θα χρησιμοποιεί ένα αντικείμενο και ένα proc και θα καλεί το proc πάνω σε αυτό το αντικείμενο. Αν το proc επιστρέφει ως αποτέλεσμα false, παραιτούμαστε, διαφορετικά καλούμε το proc μαζί με το επιστρεφόμενο αντικείμενο. Συνεχίζουμε την κλήση του proc μέχρι να μας επιστρέψει την τιμή false (κάτι το οποίο κάποια στιγμή πρέπει να συμβαίνει στο πρόγραμμα αλλιώς το πρόγραμμα δεν θα καταλήγει σε αδιέξοδο). Η μέθοδος θα επιστρέφει την τελευταία επιστρεφόμενη τιμή από το proc, η οποία δεν είναι false.

```
def doUntilFalse firstInput, someProc
  input  = firstInput
  output = firstInput

  while output
    input  = output
    output = someProc.call input
  end

  input
end

buildArrayOfSquares = Proc.new do |array|
  lastNumber = array.last
  if lastNumber <= 0
    false
  else
    array.pop                    #  Take off the last number...
    array.push lastNumber*lastNumber  #  ...and replace it with its square...
    array.push lastNumber-1          #  ...followed by the next smaller number.
  end
end

alwaysFalse = Proc.new do |justIgnoreMe|
  false
end

puts doUntilFalse([5], buildArrayOfSquares).inspect
puts doUntilFalse('I\'m writing this at 3:00 am; someone knock me out!', alwaysFalse)
```

```
[25, 16, 9, 4, 1, 0]
I'm writing this at 3:00 am; someone knock me out!
```

Εντάξει, παραδέχομαι πως αυτό είναι ένα αρκετά παράξενο παράδειγμα, αλλά δείχνει πόσο διαφορετικά δρα η μέθοδος μας όταν της δίνονται πολύ διαφορετικά μεταξύ τους procs.

Η μέθοδος inspect, μοιάζει αρκετά με τη μέθοδο to_s εκτός από το οτι το επιστρεφόμενο

αλφαριθμητικό προσπαθεί να σας δείξει τον κώδικα της Ruby για τη δημιουργία του αντικειμένου στο οποίο τη μεταβιβάσατε. Έτσι στο συγκεκριμένο παράδειγμα μας δείχνει όλο τον πίνακα που σπιτρέφεται από την πρώτη κλήση του doUntilFalse .Μπορεί να προσέξατε οτι στην πραγματικότητα, ποτέ δεν τετραγωνίσατε εκείνο το 0 στο τέλος του πίνακα, αλλά από τη στιγμή που το τετραγωνισμένο 0 παραμένει 0, τότε δεν χρειάζεται να το κάνετε. Και από τη στιγμή που το alwaysFalse ήταν όπως ξέρετε, πάντα λάθος,το doUntilFalse δεν έκανε τίποτα τελικά τη δεύτερη φορά που το καλέσαμε. Απλώς επέστρεψε αυτό που του μεταβιβάστηκε.

**Μέθοδοι οι οποίοι επιστρέφουν Procs**

Ένα από τα άλλα ενδιαφέροντα πράγματα που μπορείτε να κάνετε με τα procs είναι να τα δημιουργήσετε μέσα σε μεθόδους και να τα επιστρέψετε. Αυτή η δυνατότητα επιτρέπει όλα τα είδη των αλλόκοτων προγραμματιστικών δυνάμεων (πράγματα με εντυπωσιακά ονόματα, όπως τεμπέλικη αξιολόγηση, άπειρες δομές δεδομένων και δέψη/ξύστρισμα), αλλά είναι γεγονός, οτι σχεδόν ποτέ δεν υλοποιούνται αυτά στην πράξη, ούτε μπορώ να θυμηθώ να έχω δει κάποιον άλλο να τα χρησιμοποιεί αυτά στον κώδικα του. Νομίζω πως είναι το είδος των πραγμάτων, στα οποία δεν καταλήγει κανείς ως λύση για να υλοποιήσει κάτι στην Ruby, ή ίσως η Ruby σας ενθαρρύνει να χρησιμοποιήσετε άλλες λύσεις. Δεν ξέρω, σε κάθε περίπτωση, απλώς θα τα αναφέρω συνοπτικά.

Σε αυτό το παράδειγμα, η compose, δέχεται δύο procs και επιστρέφει μια νέα proc, η οποία, όταν καλείται, καλεί την πρώτη proc και μεταβιβάζει το αποτέλεσμα της στιν δεύτερη proc.

```
def compose proc1, proc2
  Proc.new do |x|
    proc2.call(proc1.call(x))
  end
end

squareIt = Proc.new do |x|
  x * x
end

doubleIt = Proc.new do |x|
  x + x
end

doubleThenSquare = compose doubleIt, squareIt
squareThenDouble = compose squareIt, doubleIt

puts doubleThenSquare.call(5)
puts squareThenDouble.call(5)
```

```
100
50
```

Σημειώστε ότι η κλήση για το proc1, πρέπει να γίνεται μέσα στην παρένθεση του proc2 έτσι ώστε αυτή να γίνεται πρώτη.

**Passing Blocks (Not Procs) into Methods**

Μεταβιβάζοντας Blocks (Όχι Procs) μέσα στις Μεθόδους.

Εντάξει, αυτό είναι ένα είδος χωρίς πρακτικό ενδιαφέρον, αλλά επίσης και κατά κάποιο τρόπο είναι ταλαιπωρία να χρησιμοποιηθεί. Ένα μεγάλο μέρος του προβλήματος είναι οτι υπάρχουν τρία βήματα, τα οποία πρέπει να διανύσετε (καθορισμός της μεθόδου, δημιουργία του proc και κλήση της μεθόδου μαζί με το proc) όταν κατά κάποιο τρόπο αισθάνεστε πως θα πρέπει να υπάρχουν μόνο δύο (ορίζοντας τη μέθοδο, και περνώντας το δικαίωμα μπλοκ στην μέθοδο, χωρίς τη χρήση proc καθόλου), δεδομένου ότι τις περισσότερες φορές δεν θέλετε να χρησιμοποιήσετε το proc / μπλοκ, αφού το μεταβιβάσετε στη μέθοδο. Λοιπόν, αν και δεν το γνωρίζετε η Ruby έχει σκεφτεί τα πάντα για μας! Στην πραγματικότητα, κάνετε ακριβώς αυτό το πράγμα κάθε φορά που χρησιμοποιείτε Επαναλήπτες (Iterators).

Θα σας δείξω ένα γρήγορο παράδειγμα πρώτα και στη συνέχεια θα το συζητήσουμε:

```
class Array

  def eachEven(&wasABlock_nowAProc)
    isEven = true  #  We start with "true" because arrays start with 0, which is even.

    self.each do |object|
      if isEven
        wasABlock_nowAProc.call object
      end

      isEven = (not isEven)  #  Toggle from even to odd, or odd to even.
    end
  end

end

['apple', 'bad apple', 'cherry', 'durian'].eachEven do |fruit|
  puts 'Yum!  I just love '+fruit+' pies, don\'t you?'
end

#  Remember, we are getting the even-numbered elements
#  of the array, all of which happen to be odd numbers,
#  just because I like to cause problems like that.
[1, 2, 3, 4, 5].eachEven do |oddBall|
  puts oddBall.to_s+' is NOT an even number!'
end
```

```
Yum!  I just love apple pies, don't you?
Yum!  I just love cherry pies, don't you?
1 is NOT an even number!
3 is NOT an even number!
5 is NOT an even number!
```

Έτσι για να μεταβιβάσετε ένα μπλοκ στην eachEven, το μόνο που είχατε να κάνετε ήταν να

κολλήσετε το μπλοκ μετά από τη μέθοδο. Μπορείτε να περάσετε ένα μπλοκ σε οποιαδήποτε μέθοδο 'αυτόν τον τρόπο, αν και πολλές μέθοδοι θα αγνοήσουν απλά το μπλοκ. Για να κάνετε τη μέθοδο σας να μην αγνοεί το μπλοκ, αλλά να το "αρπάξει" και να το μετατρέψει σε proc, βάλτε το όνομα του proc στο τέλος της λίστας παραμέτρων της μεθόδου σας, πριν από ένα συμπλεκτικό (&). Έτσι, αυτό το σημείο είναι λίγο δύσκολο, αλλά δεν είναι πολύ κακό, και πρέπει να το υλοποιήσετε μόνο μία φορά (κατά τον ορισμό της μεθόδου). Στη συνέχεια, μπορείτε να χρησιμοποιήσετε τη μέθοδο ξανά και ξανά, όπως τις ενσωματωμένες μεθόδους που λαμβάνουν μπλοκ, όπως το each και το  times (Θυμηθείτε 5.times do....).

Αν μπερδεύεστε, απλά θυμηθείτε, τι υποτίθεται οτι πράττει το eachEven :  καλεί το block, το οποίο μεταβιβάζεται μαζί με κάθε άλλο στοιχείο στον πίνακα. Από τη στιγμή που θα το γράψετε μια φορά και δουλεύει σωστά, δεν χρειάζεται να σκέφτεστε σχετικά με το τι πραγματικά κάνει  κάτω από την "κουκούλα" ("ποιο block καλείται και πότε;;"). Στην πραγματικότητα γι' αυτό γράφουμε τις μεθόδους με αυτόν τον τρόπο: έτσι ώστε να μην χρειάζεστε να σκεφτούμε ξανά το πως δουλεύουνε. Εμείς απλά τις χρησιμοποιούμε.

Θυμάμαι μια φορά που ήθελα να έχω τη δυνατότητα να χρονομετρώ, πόσο διαρκούσαν διαφορετικές ενότητες ενός προγράμματος. (Αυτό είναι επίσης γνωστό και ως σκιαγράφηση του κώδικα.) Έτσι, έγραψα μια μέθοδο η οποία συλλαμβάνει την ώρα  πριν από την εκτέλεση του κώδικα, μετά τον εκτελεί, στη συνέχεια συλλαμβάνει το χρόνο και πάλι στο τέλος και υπολογίζει τη διαφορά. Δεν μπορώ να βρω τον κώδικα τώρα, αλλά εγώ δεν το χρειάζομαι .Θα πρέπει να ήταν μάλλον κάτι σαν αυτό:

```
def profile descriptionOfBlock, &block
  startTime = Time.now

  block.call

  duration = Time.now - startTime

  puts descriptionOfBlock+':  '+duration.to_s+' seconds'
end

profile '25000 doublings' do
  number = 1

  25000.times do
    number = number + number
  end

  puts number.to_s.length.to_s+' digits'  #  That is, the number of digits in this HUGE number.
end

profile 'count to a million' do
  number = 0

  1000000.times do
    number = number + 1
  end
end
```

```
7526 digits
25000 doublings:  0.246768 seconds
count to a million:  0.90245 seconds
```

Πόσο απλό! Πόσο κομψό! Με αυτή την μικροσκοπική μέθοδο, μπορώ τώρα εύκολα να χρονομετρώ οποιοδήποτε τμήμα, οποιουδήποτε προγράμματος θέλω. Απλά ρίχνω τον κώδικα σε ένα μπλοκ και το στέλνω στο profile Τι θα μπορούσε να είναι απλούστερο; Στις περισσότερες γλώσσες, θα είχα να προσθέσω ρητά αυτόν τον κώδικα χρονισμού (τα πράγματα στο profile) γύρω από κάθε σημείο το οποίο ήθελα να χρονομετρήσω.. Στη Ruby, ωστόσο, καταφέρνω να τα κρατήσω όλα σε ένα μέρος, και (το σημαντικότερο) μακρυά από το δρόμο μου!

**Ορισμένα πράγματα που είναι καλό να δοκιμάσετε:**

- Το Ρολόι του Παππού.  Γράψτε μια μέθοδο η οποία παίρνει ένα Block και το καλεί μία φορά για κάθε ώρα που περνάει σήμερα. Με αυτό τον τρόπο, αν  εω έπρεπε να μεταβιβάσω στο block: do puts 'DONG!' end , θα χτυπούσε όπως το ρολόι του παππού. Ελέγξτε τη μέθοδο σας με μερικά διαφορετικά blocks (συμπεριλαμβανομένου εκείνου που μόλις σας έδωσα ). **Συμβουλή:** Μπορείτε να χρησιμοποιήσετε Time.now.hour για να πάρει την τρέχουσα ώρα. Ωστόσο, αυτή επιστρέφει έναν αριθμό μεταξύ 0 και 23, οπότε θα πρέπει να αλλάξει αυτούς τους αριθμούς, προκειμένου να πάρει τους συνήθεις αριθμούς ενός ρολογιού (1 έως 12).
- Πρόγραμμα Καταγραφέας. Γράψτε μια μέθοδο που ονομάζεται log, το οποίο λαμβάνει μια αλφαριθμητική περιγραφή ενός μπλοκ και, φυσικά, ένα μπλοκ. Παρόμοια με την doSelfImportantly, θα πρέπει να εμφανίζει ένα αλφαριθμητικό λέγοντας ότι έχει αρχίσει το μπλοκ, και ένα άλλο αλφαριθμητικό στο τέλος να σας ενημερώνει ότι έχει τελειώσει το μπλοκ, αλλά και σας λέει ποιο είναι το μπλοκ, που  επέστρεψε. Δοκιμάστε τη μέθοδο σας, στέλνοντας της ένα μπλοκ κώδικα. Μέσα στο μπλοκ, βάλτε μια άλλη κλήση στο log, μεταβιβάζοντας ένα άλλο μπλοκ σε αυτό. (. Αυτό ονομάζεται ωοτοκία). Με άλλα λόγια, το αποτέλεσμα  σας θα είναι κάπως έτσι:

```
Beginning "outer block"...
Beginning "some little block"...
..."some little block" finished, returning:  5
Beginning "yet another block"...
..."yet another block" finished, returning:  I like Thai food!
..."outer block" finished, returning:  false
```

• Καλύτερος Καταγραφέας. Το αποτέλεσμα από τον προηγούμενο Κταγραφέα, ήταν κάπως δύσκολο να διαβαστεί και θα γινόταν τόσο χειρότερο, όσο πειρσσότερο το χρησιμοποιούσατε.  Θα ήταν τόσο ευκολότερο να το διαβάσετε αν χρησιμοποιούσατε εσοχές στις γραμμές στα εσωτερικά μπλοκ. Για να το πετύχετε αυτό θα χρειάζεστε να παρακολουθείτε του πόσο βαθιά φωλιασμένοι θα είστε κάθε φορά που ο Καταγραφέας  θέλει να γράψει κάτι. Για να το κάνετε αυτό, χρησιμοποιήστε μια καθολική μεταβλητή, μια μεταβλητή την οποία μπορείτε να “δείτε” (να χειριστείτε, να αποκτήσετε πρόσβαση) , από οπουδήποτε μέσα στον κώδικα σας. It would be so much easier to read if it indented the lines in the inner blocks. . Για να δημιουργήσετε μια καθολική μεταβλητή, απλώς φροντίστε να προηγηθεί του ονόματος της ο χαρακτήρας $, όπως στις παρακάτω: $global, $nestingDepth, and $bigTopPeeWee. Στο τέλος ο Καταγραφέας σας, θα έπρεπε να περιλαμβάνει τον παρακάτω κώδικα:

```
Beginning "outer block"...
  Beginning "some little block"...
    Beginning "teeny-tiny block"...
    ..."teeny-tiny block" finished, returning:  lots of love
  ..."some little block" finished, returning:  42
  Beginning "yet another block"...
  ..."yet another block" finished, returning:  I love Indian food!
..."outer block" finished, returning:  true
```

Λοιπόν αυτά είναι όλα, τα οποία πρόκειται να μάθετε από αυτό το εγχειρίδιο. Συγχαρητρήρια. Μάθατε πολλά. Μπορεί να μην νιώθετε σαν να θυμόσαστε τα πάντα, ή μπορεί να παραλείψατε κάποια τμήματα... αλήθεια όμως, δεν πειράζει, αρκούν όσα μάθατε. Το να προγραμματίζετε, δεν έχει να κάνει με το τι γνωρίζετε, έχει να κάνει με το τι μπορείτε να καταλάβετε. . Εφ 'όσον ξέρετε πού να βρείτε τα πράγματα που ξεχάσατε ή παραλείψατε, τότε είστε μια χαρά. Ελπίζω να μην νομίζετε ότι έγραψα όλα αυτά χωρίς να ψάχνω να τα βρω συνεχώς! Επειδή ακριβώς αυτό έκανα. Πήρα επίσης πολλή βοήθεια με τον κώδικα που τρέχει όλα τα παραδείγματα σε αυτό το σεμινάριο. Αλλά όταν ήμουν ψάχνουν πράγματα επάνω, και ο οποίος μου ζητώντας βοήθεια; Επιτρέψτε μου να σας δείξω ...As long as you know where to find out the things you forgot, you're doing just fine.Επίσης έλαβα πολύ βοήθεια με τον κώδικα που τρέχει όλα τα παραδείγματα σε αυτό το εγχειρίδιο. Αλλά που έψαχνα να βρω τη βοήθεια; και σε ποιον απευθυνόμουν γι αυτή; . Αφήστε με να σας δείξω.

## 11. Πέρα από αυτό το εγχειρίδιο

Λοιπόν που πβαδίζουμε τώρα; Αν έχετε ένα ερώτημα, μια απορία, ποιον μπορείτε να ρωτήσετε; Τι θα κάνετε αν θέλετε το πρόγραμμα σας να ανοίγει μια ιστοσελίδα, να στείλει ένα μήνυμα ηλεκτρονικού ταχυδρομείου, ή να αλλάξει το μέγεθος μιας ψηφιακής φωτογραφίας; Λοιπόν, υπάρχουν πάρα πολλά μέρη για να βρείτε βοήθεια σχετικά με τη Ruby. Δυστυχώς αυτό σας φαίνεται κάπως γενικό και καθόλου βοηθητικό, μάλλον. Για μενα, υπάρχουν πραγματικά μόνο τρία μέρη στα οποίο ψάχνω για βοήθεια σχετικά με τη Ruby. Αν είναι ένα μικρό ερώτημα και νομίζω πως μπορώ να πειραματιστώ μόνος μας για να βρω την απάντηση, τότε χρησιμοποιώ την irb (Interactive Ruby). Αν είναι ένα μεγαλύτερο ερώτημα, το αναζητώ στο Pickaxe. Τέλος αν απλά δεν μπορώ να το καταλάβω μόνος μου, τότε ζητάω βοήθεια στο ruby-talk

### IRB: Interactive Ruby

Αν εγκαταστήσατε τη Ruby, τότε εγκαταστήσατε και την irb. Για να την χρησιμοποιήσετε, απλά πηγαίντε στη γραμμή εντολών και πληκτρολογήστε: irb. Όταν βρίσκετστε μέσα στην irb, μπορείτε να πληκτρολογήσετε, οποιαδήποτε έκφραση της Ruby, θέλετε, και θα σας επιστρέψει την τιμή της. Πληκτρολογήστε 1 + 2, και θα σας επιστρέψει 3. (Σημειώστε οτι δεν απαιτείται να χρησιμοποιήστε την puts). Είναι κάτι σαν μια γιγαντιαία αριθμομηχανή Ruby. Όταν τελειώσετε απλά πληκτρολογήστε exit.

Υπάρχουν πολλά περισσότερα στην irb από αυτό, αλλά μπορείτε να μάθετε σχετικά με αυτά στο pickaxe.

**The Pickaxe: "Programming Ruby"**

Απολύτως το βιβλίο Ruby που χρειάζεται κάποιος να πάρει είναι "Programming Ruby, The Pragmatic Programmer's Guide", του David Thomas και Andrew Hunt the Pragmatic Programmers). Αν και συστήνω ιδιαίτερα να επιλέξετετη 2η έκδοση αυτού του εξαιρετικού βιβλίου, με όλες τις τελευταίες αλλαγές της Ruby να καλύπτονται. Μπορείτε επίσης να πάρετε μια ελαφρώς παλιότερη (αλλά ως επί το πλείστον σχετική) έκδοση δωρεάν on-line. (Στην πραγματικότητα, εάν έχετε εγκαταστήσει την έκδοση για Windows της Ruby, τότε την έχετε ήδη.) Μπορείτε να βρείτε σχεδόν τα πάντα σχετικά με τη Ruby, από τα βασικά μέχρι τα πιο προχωρημένα, σε αυτό το βιβλίο. Είναι εύκολο να το διαβάσετε, είναι περιεκτικό. Είναι απλώς τέλειο. Εύχομαι κάθε γλώσσα προγραμματισμού να είχε ένα βιβλίο τέτοιας ποιότητας. Στο πίσω μέρος του βιβλίου, θα βρείτε μια τεράστια ενότητα με λεπτομέρειες για κάθε μέθοδο σε κάθε τάξη, εξηγώντας τη και δίνοντας παραδείγματα. Αγαπώ αυτό το βιβλίο! Υπάρχουν πολλά διαφορετικά μέρη από τα οποία μπορείτε να το πάρετε (συμπεριλαμβανομένης και της ιστοσελίδας των Pragmatic Programmers), αλλά το αγαπημένο μου μέρος είναι στο ruby-doc.org. Αυτή η εκδοχή έχει ένα ωραίο πίνακα περιεχομένων στην άκρη, καθώς και ευρετήριο. (Το ruby-doc.org έχει σπουδαίες άλλες τεκμηριώσεις και εγχειρίδια , όπως για το Core API και την πρότυπη βιβλιοθήκη -Standard Library- ... βασικά, τεκμηριώνει οποιοδήποτε χαρακτηριστικό και ιδιότητα προστίθεται στη Ruby κατ' ευθείαν από το κουτί. Ελέγξτε το.)

Και γιατί ονομάζεται «η αξίνα" (The Pickaxe); Λοιπόν, υπάρχει μια εικόνα μιας αξίνας στο εξώφυλλο του βιβλίου. Είναι ένα χαζό όνομα, υποθέτω, αλλά κόλλησε.

.

**Ruby-Talk:Η Mailing List της Ruby**

Ακόμα και με την irb και το pickaxe (Αξίνα), μερικές φορές δεν θα μπορείτε να καταλάβετε κάτι ή ίσως θέλετε να γνωρίζετε αν κάποιος έκανε ήδη κάτι, στο οποίο εσείς εργάζεστε και θέλετε να διαπιστώσετε αν θα μπορούσατε να το χρησιμοποιήσετε. Σε αυτές τις περιπτώσεις, το μέρος που θα έπρεπε να επισκεφτείτε είναι το ruby-talk, η λίστα ηλεκτρονικού ταχυδρομείου της Ruby. Είναι γεμάτη από φιλικούς, έξυπνους και πρόθυμους να βοηθήσοτν ανιρώπους. Για να μάθετε πρισσότερα σχετικά με αυτή ή να εγγραφείτε σε αυτή κοιτάξτε εδώ.

**ΠΡΟΕΙΔΟΠΟΙΗΣΗ:** Υπάροχνυ πάρα πολλά μηνύματα καθημερινώς στην λίστα ηλεκτρονικού ταχυδρομείου. Έχω ρυθμίσει τον λογαριασμό μου, ώστε αυτόματα να κατευθύνονται σε έναν ξεχωριστό φάκελο, ώστε να μην με ενοχλούν. Αν δεν θέλετε να μπλέκεστε με όλα αυτά τα μηνύματα, τότε δεν χρειάζεται να το κάνετε. Η λίστα ηλεκτρονικού ταχυδρομείου ruby-talk "καθρεφτίζεται" στο newsgroup comp.lang.ruby, και το αντίστροφο, έτσι μπορείτε να δείτε τα ίδια μηνύματα εκεί. Έτσι ή αλλιώς βλέπετε τα ίδια μηνύματα, αλλά απλώς σε μία ελαφρώς διαφορετική μορφή.

**Tim Toady**

Υπάρχει κάτι από το οποίο έχω προσπαθήσει να σας προστατεύσω, αλλά στο οποίο σίγουρα θα τρέξετε σύντομα, είναι η έννοια της TMTOWTDI (προφέρεται "Tim Toady»): Υπάρχουν περισσότεροι από ένας τρόποι να το κάνετε (There's More Than One Way To Do It ). Τώρα κάποιοι θα σας πουν τι θαυμάσιο πράγμα είναι το TMTOWTDI, ενώ κάποιοι άλλοι αισθάνονται αρκετά διαφορετικά. Δεν έχω πραγματικά ισχυρά συναισθήματα για αυτό εν γένει, αλλά νομίζω ότι είναι ένας άσχημος τρόπος για να διδάξεις κάποιον πώς να προγραμματίζει. (Λες και η μάθηση ενός και μόνου τρόπου για να κάνετε κάτι δεν ήταν αρκετά προκλητική και δύσκολη!)

Ωστόσο, τώρα που κινείστε πέρα από αυτό το εχγειρίδιο, θα συναντήσετε πολύ πιο διαφοροποιημένο κώδικα. Για παράδειγμα, μπορώ να σκεφτώ τουλάχιστον άλλους πέντε τρόπους για να δημιουργήσετε ένα αλφαριθμητικό (εκτός από το να περιβάλλεται κάποιο κείμενο με μονά εισαγωγικά), και ο καθένας λειτουργεί λίγο διαφορετικά. Εγώ σας έδειξα μόνο τον απλούστερο από τους έξι τρόπους. Και όταν μιλήσαμε για διακλάδωση, σας έδειξα το if, αλλά δεν σας έδείξα το unless. Θα σας αφήσω να το ανακαλύψετε αυτό στην IRB.
Μια άλλη καλή, μικρή συντόμευση που μπορείτε να χρησιμοποιήσετε με τα if, unless, και while, είναι η χαριτωμένη εκδοχή της μίας γραμμής:

```
#  These words are from a program I wrote to generate
#  English-like babble.  Cool, huh?
puts 'grobably combergearl kitatently thememberate' if 5 == 2**2 + 1**1
puts 'enlestrationshifter supposine follutify blace' unless 'Chris'.length == 5
```

```
grobably combergearl kitatently thememberate
```

Και τελικά, υπάρχει ένας άλλος τρόπος γραφής μεθόδων, ο οποίος δέχεται blocks (όχι procs). Είδαμε το παράδειγμα, όπου πήραμε το block και το μετατρέψαμε σε proc, χρησιμοποιώντας το κόλπο &block, στην λίστα των παραμέτρων σας, όταν ορίζετε τη συνάρτηση (function) .Στη συνέχεια για να καλέσετε το block, απλώς χρησιμοποιείτε την κλήση του block. Λοιπόν υπάρχει ένας συντομότερος δρόμος (Αν και προσωπικά των βρίσκω πιο μπερδεμένο) . Αντί γι' αυτό:

```
def doItTwice(&block)
  block.call
  block.call
end

doItTwice do
  puts 'murditivent flavitemphan siresent litics'
end
```

```
murditivent flavitemphan siresent litics
murditivent flavitemphan siresent litics
```

Μπορείτε να κάνετε αυτό:

```
def doItTwice
  yield
  yield
end
```

```
doItTwice do
  puts 'buritiate mustripe lablic acticise'
end
```

```
buritiate mustripe lablic acticise
buritiate mustripe lablic acticise
```

Χμ, δεν ξέρω... τι λέτε εσείς;Μπορεί να είμαι εγώ ο παράξενος, αλλά... yield; Αν ήταν κάτι σαν αυτό:  call_the_hidden_block ή τέλος πάντων κάτι που θα έβγαζε περισσότερο νόημα. Για πολλούς ανθρώπους το να λένε yield έχει νόημα. Αλλά υποθεω πως γι' αυτό υπάρχει το  TMTOWTDI τελικά: Αυτοί το κάνουν με τον τρόπο τους και εγώ θα το κάνω με τον δικό μου τρόπο.

**ΤΕΛΟΣ**

Χρησιμοποιήστε το για καλό και όχι για κακό. :-) και αν βρήκατε αυτό το εγχειρίδιο χρήσιμο (ή μπερδεμένο ή αν βρήκατε κάποιο λάθος), ενημερώστε με