RUBY / ΕΛΛΗΝΙΚΗ ΕΚΔΟΣΗ

Λύσεις και εξηγήσεις

208 μαθήματα σύνταξης και concepts

Από τον καθημερινό κώδικα στις πιο ειδικές τεχνικές.

RUBY

```ruby
names
  .map(&:strip)
  .reject(&:empty?)
  .tap { |cleaned| audit << cleaned.length }
  .then { |cleaned| cleaned.join(", ") }
```

Ruby 3.4 / έκδοση 12.09.2026

Πριν αρχίσεις

Κάθε νέο μάθημα εισάγει μια βασική ιδέα μέσα από μικρή πρακτική δουλειά. Οι κανόνες δίνονται. Ο στόχος είναι να γράφεις και να διαβάζεις Ruby με άνεση.

Διάβασε το παράδειγμα, πρόβλεψε την επιστροφή, γράψε την προσπάθειά σου και τρέξε τα tests. Η παραλλαγή χρησιμοποιεί την ίδια γνώση σε λίγο διαφορετικό αίτημα.

Αυτό το τεύχος περιέχει τις λύσεις, τις εξηγήσεις και τις απαντήσεις των παραλλαγών. Το ruby-lessons.pdf περιέχει τα μαθήματα και τις εκφωνήσεις.

```sh
cd syntax-first/ruby
bin/lesson 014
bin/lesson 014 --solution
bin/lesson 014 --examples
```

Η πρώτη εντολή ελέγχει την προσπάθεια, η δεύτερη τη χωριστή λύση και η τρίτη τα παραδείγματα. Οι άλυτες προσπάθειες αποτυγχάνουν εσκεμμένα. Τα εργαλεία χρησιμοποιούν την ήδη εγκατεστημένη Ruby 3.4 και RSpec 3.13.2.

Οι μεγάλες γραμμές κώδικα αναδιπλώνονται οπτικά στο PDF. Το > στο περιθώριο σημαίνει συνέχεια της ίδιας γραμμής. Για ακριβή αντιγραφή κώδικα χρησιμοποίησε τα συνοδευτικά .rb αρχεία.

Περιεχόμενα

Προτεραιότητα 1: Καθημερινός κορμός

Βασική σύνταξη και επιστροφή αποτελέσματος

001. Εκτέλεση και παρατήρηση · Γέφυρα
002. Τιμές και εκφράσεις · Γέφυρα
003. Κείμενο με δεδομένα · Γέφυρα
004. Μέθοδος που επιστρέφει αποτέλεσμα · Γέφυρα
005. Επιλογή αποτελέσματος · Γέφυρα
006. Τι θεωρείται αληθές
007. Σύνδεση κανόνων
008. Πολλές ονομασμένες περιπτώσεις
009. Έξοδος νωρίς
010. Κανόνες αποστολής · Εμπέδωση

Arrays, blocks και οι βασικές πράξεις συλλογών

011. Διατεταγμένες συλλογές
012. Συγκέντρωση τιμών
013. Ένα block για κάθε στοιχείο
014. Μία νέα τιμή για κάθε παλιά
015. Κράτησε όσα ταιριάζουν
016. Αφαίρεσε όσα απορρίπτονται
017. Μετατροπή ή φίλτρο; · Εμπέδωση
018. Το πρώτο που ταιριάζει
019. Ερωτήσεις για όλη τη συλλογή
020. Πλήθος με κριτήριο
021. Άθροισμα με μετατροπή
022. Θέση μαζί με τιμή
023. Μικρό βαθμολόγιο · Εμπέδωση

Καθημερινά strings και μεταβολή τιμών

024. Καθάρισμα άκρων
025. Πεζά και κεφαλαία
026. Κείμενο σε πεδία
027. Πεδία σε κείμενο
028. Διαστήματα τιμών
029. Κομμάτια από μια τιμή
030. Απλή αναζήτηση κειμένου
031. Αντικατάσταση σταθερού κειμένου
032. Χαρακτήρες και γραμμές
033. Καθάρισμα λίστας ονομάτων · Εμπέδωση
034. Μεταβολή του ίδιου αντικειμένου

035. Δύο ονόματα, ένα αντικείμενο
036. Καθαρές ετικέτες συμμετοχής · Εμπέδωση
Hashes και πρόσβαση σε δεδομένα
037. Εγγραφές με ονομασμένα πεδία
038. Λείπει κλειδί ή υπάρχει τιμή nil;
039. Κλειδί και τιμή στο block
040. Δεδομένα μέσα σε δεδομένα
041. Συγχώνευση ρυθμίσεων
042. Προεπιλογές εγγραφής · Εμπέδωση
043. Μετατροπή κλειδιών
044. Μετατροπή τιμών
045. Επιλογή πεδίων
046. Λίστα παραγγελιών · Εμπέδωση
Ορίσματα, scope και προαιρετικές τιμές
047. Προαιρετικά ορίσματα
048. Ονομασμένα ορίσματα
049. Όρια ορατότητας
050. Προαιρετικός receiver
051. Ανάθεση υπό συνθήκη
052. Μικρές διορθώσεις syntax · Εμπέδωση
Οι πιο χρήσιμες μετατροπές και συνόψεις
053. Ταξινόμηση έτοιμων τιμών
054. Ταξινόμηση με κλειδί
055. Μικρότερο και μεγαλύτερο
056. Μία εμφάνιση ανά ταυτότητα
057. Αφαίρεση απουσίας
058. Ομάδες με κοινό γνώρισμα
059. Συχνότητες τιμών
060. Μικρές ομάδες προϊόντων · Εμπέδωση
061. Δύο συμπληρωματικές ομάδες
062. Μετατροπή μαζί με απόρριψη
063. Πολλά αποτελέσματα από κάθε στοιχείο
064. Παρατήρηση μέσα σε αλυσίδα
065. Συνέχεια με το αποτέλεσμα του block
066. Ποια μέθοδος ταιριάζει; · Εμπέδωση
067. Αναφορά αποθήκης · Εμπέδωση
Κλάσεις, αντικείμενα και συνεργασία κώδικα
068. Κατάσταση μαζί με συμπεριφορά
069. Ανάγνωση και αλλαγή ιδιοτήτων
070. Ο τρέχων receiver
071. Ονόματα με συμβάσεις

072. Δημόσιο API και βοηθητικές μέθοδοι
073. Μέθοδοι της κλάσης
074. Ονόματα σε δικό τους χώρο
075. Δύο μικρά αντικείμενα · Εμπέδωση
076. Αντικείμενα που συνεργάζονται
077. Εξειδίκευση υπάρχουσας συμπεριφοράς
078. Συνέχεια στη γονική υλοποίηση
079. Κοινή συμπεριφορά με module
080. Μικρός κατάλογος κρατήσεων · Εμπέδωση

Έλεγχος δεδομένων και διαχείριση σφαλμάτων

081. Απόρριψη άκυρης κλήσης
082. Χειρισμός συγκεκριμένου σφάλματος
083. Το σφάλμα ως αντικείμενο
084. Τιμή από επιτυχία ή αποτυχία
085. Μικρός πίνακας αποτελεσμάτων · Εμπέδωση
086. Ενέργεια που γίνεται πάντα
087. Δικό σου είδος αποτυχίας
088. Απουσία αποτελέσματος χωρίς εξαίρεση
089. Μικρός εισαγωγέας εγγραφών · Εμπέδωση

Tests, αρχεία, JSON και μικρά εργαλεία

090. Κώδικας σε περισσότερα αρχεία
091. Βιβλιοθήκες και εξαρτήσεις
092. Γράψε δικό σου παράδειγμα συμπεριφοράς
093. Έλεγχος αποτυχίας και μεταβολής
094. Ανάγνωση ολόκληρου κειμένου
095. Άνοιγμα με ορισμένο χρόνο ζωής
096. Γραμμές μία μία
097. Παραγωγή αρχείου
098. Διαδρομές και επιλογή αρχείων
099. Ορίσματα από το Terminal
100. Ροές εισόδου και εξόδου
101. JSON
102. CSV
103. Ρυθμίσεις από το περιβάλλον
104. Αναφορά από fixtures · Εμπέδωση
105. Τοπικό εργαλείο αναφορών · Εμπέδωση

Προτεραιότητα 2: Χρήσιμη συνέχεια

Δικά σου blocks και callable αντικείμενα

106. Η δική σου μέθοδος δέχεται block
107. Το block είναι προαιρετικό

108. Block ως τιμή
109. Μεταβλητές που θυμάται ένα block
110. Συνάρτηση με δικά της όρια
111. Από πού επιστρέφει το return;
112. Callable με προσαρμογή · Εμπέδωση
113. Υπάρχουσα μέθοδος ως callable
114. Μετατροπή σε block
115. Μια αλυσίδα που εξηγείται · Εμπέδωση

Splat, keywords και αποσύνθεση τιμών

116. Μεταβλητό πλήθος ορισμάτων
117. Άνοιγμα λίστας στην κλήση
118. Ονομασμένες επιλογές ως hash
119. Η ίδια μέθοδος με διαφορετικές κλήσεις · Εμπέδωση
120. Ανάθεση σε πολλά ονόματα
121. Δομή παραμέτρων block
122. Formatter με επιλογές · Εμπέδωση

Συγκέντρωση αποτελεσμάτων και προεπιλογές

123. Προεπιλεγμένη τιμή hash
124. Προεπιλογή που κατασκευάζεται
125. Αποτέλεσμα που ενημερώνεται
126. Αποτέλεσμα που περνά στο επόμενο βήμα
127. Ανεξάρτητες αρχικές τιμές
128. Μνήμη προηγούμενου αποτελέσματος

Έλεγχος επανάληψης και κομμάτια συλλογών

129. Έλεγχος της επανάληψης
130. Επανάληψη με συνθήκη
131. Συγκεκριμένα βήματα επανάληψης
132. Αρχή και υπόλοιπο
133. Ομάδες σταθερού μεγέθους
134. Διαδοχικά ζευγάρια
135. Συλλογές δίπλα δίπλα
136. Σελίδες και γειτονικές τιμές · Εμπέδωση
137. Πρόγραμμα ομάδων · Εμπέδωση

Χρήση υπάρχοντος Enumerator

138. Η επανάληψη ως αντικείμενο
139. Ζήτησε το επόμενο στοιχείο
140. Κοίτα χωρίς να προχωρήσεις
141. Προσθήκη θέσης σε μια ακολουθία
142. Πρόβλεψη της επόμενης τιμής · Εμπέδωση

Regular expressions για καθημερινά δεδομένα

143. Έλεγχος μορφής κειμένου

144. Ομάδες και επαναλήψεις σε regex
145. Εξαγωγή ονομασμένων πεδίων
146. Πολλά ταιριάσματα
147. Αντικατάσταση με υπολογισμό
148. Κείμενο σε δομημένες τιμές · Εμπέδωση
Ημερομηνίες, σύνολα και εργαλεία του Terminal
149. Χρονικές στιγμές και διάρκειες
150. Ημερολογιακές ημερομηνίες
151. Σύνολα τιμών
152. Ακριβείς δεκαδικές τιμές
153. Επιλογές εντολής
154. Εκτέλεση άλλου τοπικού προγράμματος
155. Μετατροπή τοπικών δεδομένων · Εμπέδωση
Ισότητα, σταθερές τιμές και μικρές εγγραφές
156. Προστασία από μεταβολή
157. Ισότητα και ταυτότητα
158. Μικρή εγγραφή χωρίς πολύ boilerplate
159. Τιμή με σταθερά μέλη
160. Συμπεριφορά πάνω σε ένα αντικείμενο

Προτεραιότητα 3: Πιο ειδικές ανάγκες

Πρόσθετες μορφές σύνταξης και κειμένου
161. Σύντομες μορφές εκφράσεων
162. Σύντομες παράμετροι block
163. Σε ποια κλήση ανήκει το block;
164. Προώθηση ορισμάτων και block
165. Μορφοποίηση τιμών
166. Άλλες μορφές literals
167. Χαρακτήρας, byte και encoding
Pattern matching δομών
168. Ταίριασμα δομής πίνακα
169. Ταίριασμα δομής εγγραφής
170. Περιορισμοί μέσα στο pattern
171. Ανάθεση μέσω pattern
172. Pattern matching δικών σου αντικειμένων
173. Μικρός δρομολογητής μηνυμάτων · Εμπέδωση
Δικές σου ακολουθίες και lazy επεξεργασία
174. Δική σου παραγωγή τιμών
175. Μέθοδος με και χωρίς block
176. Οι μέθοδοι συλλογών στη δική σου κλάση
177. Υπολογισμός όταν ζητηθεί

178. Σταμάτημα με αρκετά αποτελέσματα
179. Ακολουθία χωρίς έτοιμο τέλος
180. Σύνδεση και επανάληψη ακολουθιών
181. Ομάδες γειτονικών στοιχείων
182. Όσο ισχύει ο κανόνας στην αρχή
183. Γραμμές και στήλες
184. Περιστροφή σειράς
185. Ουρά που διαβάζεται σταδιακά · Εμπέδωση

Προτεραιότητα 4: Εξειδικευμένες τεχνικές

Πρωτόκολλα αντικειμένων και metaprogramming

186. Δικά σου συγκρίσιμα αντικείμενα
187. Τελεστές ως μέθοδοι
188. Μέθοδος σε ένα μόνο αντικείμενο
189. Η σειρά αναζήτησης μεθόδων
190. Επιλογή μεθόδου με όνομα
191. Ορισμός μεθόδων από δεδομένα
192. Μικρές τιμές και δυναμικές κλήσεις · Εμπέδωση
193. Αντιγραφή και αρχικοποίηση αντιγράφου
194. Ορισμός εναλλακτικού ονόματος
195. Απουσία μεθόδου ως πρωτόκολλο
196. Διαφορετικό self μέσα στο block
197. Μικρή εσωτερική γλώσσα
198. Μικρό API αναφορών · Εμπέδωση
199. Αλλαγή συμπεριφοράς σε περιορισμένο scope

Ειδική ροή εκτέλεσης και concurrency

200. Περιορισμένη επανάληψη αποτυχημένης προσπάθειας
201. Δύο ανεξάρτητες εργασίες
202. Κοινή μεταβαλλόμενη κατάσταση
203. Επικοινωνία μεταξύ threads
204. Μικρή ουρά εργασιών με έλεγχο · Εμπέδωση
205. Έξοδος από εμφωλευμένα blocks
206. Παύση και συνέχιση εργασίας
207. Πού συνεχίζει η εκτέλεση; · Εμπέδωση
208. Απομονωμένες εργασίες με μηνύματα

001 / ΛΥΣΗ

Εκτέλεση και παρατήρηση · Γέφυρα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `21`.

```ruby
price = 7
seats = 3
price * seats
```

Η κύρια άσκηση

```ruby
def ticket_preview(price, seats)
  total = price * seats
  puts total
  total
end
```

Η τελευταία γραμμή είναι `total`: έτσι επιστρέφει αριθμό. Αν τελείωνε στην `puts`, θα επέστρεφε `nil`. Η εμφάνιση και η επιστροφή είναι δύο διαφορετικές ενέργειες.

Η παραλλαγή

Υπολόγισε δύο εισιτήρια των 9 και επίστρεψε το αποτέλεσμα της `p`.

```ruby
p(9 * 2)
```

Αποτέλεσμα: `18`.

002 / ΛΥΣΗ

Τιμές και εκφράσεις · Γέφυρα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `14`.

```ruby
boxes = "4".to_i
per_box = 3
boxes * per_box + 2
```

Η κύρια άσκηση

```ruby
def remaining_seats(capacity_text, adults_text, children_text)
  capacity = capacity_text.to_i
  booked = adults_text.to_i + children_text.to_i
  capacity - booked
end
```

Μετατρέπουμε πριν από την πρόσθεση. Η πρόσθεση δύο strings ενώνει κείμενα, οπότε `"6" + "3"` δεν μετρά εννέα θέσεις.

Η παραλλαγή

Μοίρασε το έγκυρο κείμενο ποσού "7" σε δύο ίσα μέρη με δεκαδική διαίρεση.

```ruby
"7".to_f / 2
```

Αποτέλεσμα: `3.5`.

003 / ΛΥΣΗ

Κείμενο με δεδομένα · Γέφυρα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hello Mina!"`.

```ruby
name = "Mina"
"Hello #{name}!"
```

Η κύρια άσκηση

```ruby
def confirmation(name, seats, price)
  "Όνομα: #{name} | Θέσεις: #{seats} | Σύνολο: #{seats * price}"
end
```

Η πράξη βρίσκεται μέσα στο `#{...}`. Κράτησε τα κενά και τα διαχωριστικά της σύμβασης. Μονά εισαγωγικά θα άφηναν το interpolation ως απλό κείμενο.

Η παραλλαγή

Φτιάξε δύο γραμμές: στην πρώτη το όνομα Eva και στη δεύτερη τις 2 θέσεις.

```ruby
name = "Eva"
seats = 2
"#{name}\n#{seats}"
```

Αποτέλεσμα: `"Eva\n2"`.

004 / ΛΥΣΗ

Μέθοδος που επιστρέφει αποτέλεσμα · Γέφυρα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[Jo]"`.

```ruby
def badge(name)
  "[#{name}]"
end
badge("Jo")
```

Η κύρια άσκηση

```ruby
def order_total(quantity, unit_price)
  quantity * unit_price
end

def order_label(code, quantity, unit_price)
  "#{code}: #{order_total(quantity, unit_price)}"
end
```

Η `order_label` χρησιμοποιεί την επιστροφή της `order_total`. Αν προσθέσεις `puts` τελευταία, αλλάζει η επιστροφή σε `nil`.

Η παραλλαγή

Όρισε μέθοδο που προσθέτει επίθημα ! σε ένα κείμενο και κάλεσέ την για "Ετοιμο".

```ruby
def announce(text)
  "#{text}!"
end
announce("Ετοιμο")
```

Αποτέλεσμα: `"Ετοιμο!"`.

005 / ΛΥΣΗ

Επιλογή αποτελέσματος · Γέφυρα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"small"`.

```ruby
seats = 4
if seats >= 5
  "group"
else
  "small"
end
```

Η κύρια άσκηση

```ruby
def order_status(paid, quantity)
  if quantity == 0
    "κενή"
  elsif paid
    "έτοιμη"
  else
    "αναμονή"
  end
end
```

Ο ειδικός κανόνας quantity == 0 προηγείται του paid. Η σειρά των κλάδων είναι μέρος της σύμβασης.

Η παραλλαγή

Δώσε την ετικέτα "μεγάλη" από 10 θέσεις και πάνω, αλλιώς "μικρή". Δοκίμασε ακριβώς το 10.

```ruby
seats = 10
if seats >= 10
  "μεγάλη"
else
  "μικρή"
end
```

Αποτέλεσμα: `"μεγάλη"`.

006 / ΛΥΣΗ

Τι θεωρείται αληθές

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"present"`.

```ruby
value = 0
if value
  "present"
else
  "absent"
end
```

Η κύρια άσκηση

```ruby
def choice_label(value)
  if value.nil?
    "λείπει"
  elsif value == false
    "όχι"
  else
    "δόθηκε"
  end
end
```

Χρησιμοποιούμε `nil?` για απουσία. Ένας σκέτος έλεγχος `if value` θα ένωνε nil και false στον ίδιο κλάδο.

Η παραλλαγή

Επίστρεψε "default" μόνο όταν το value είναι nil, διατηρώντας το false.

```ruby
value = false
if value.nil?
  "default"
else
  value
end
```

Αποτέλεσμα: `false`.

007 / ΛΥΣΗ

Σύνδεση κανόνων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"guest"`.

```ruby
label = nil
label || "guest"
```

Η κύρια άσκηση

```ruby
def entry_allowed?(paid, banned)
  paid && !banned
end

def display_choice(value)
  value || "default"
end
```

Η `display_choice` σκόπιμα αντικαθιστά και false. Αν το false ήταν έγκυρη επιλογή που πρέπει να κρατηθεί, θα χρειαζόταν ο κανόνας nil του 006.

Η παραλλαγή

Χωρίς if, κράτησε ασφαλή την κλήση nil?: αν υπάρχει κείμενο δώσε την class του, αλλιώς nil.

```ruby
text = nil
text && text.class
```

Αποτέλεσμα: `nil`.

008 / ΛΥΣΗ

Πολλές ονομασμένες περιπτώσεις

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"open"`.

```ruby
code = "B"
case code
when "A", "B"
  "open"
else
  "closed"
end
```

Η κύρια άσκηση

```ruby
def status_message(code)
  case code
  when "new", "queued"
    "αναμονή"
  when "done"
    "έτοιμο"
  when "cancelled"
    "ακυρώθηκε"
  else
    "άγνωστο"
  end
end
```

Τα δύο κείμενα γράφονται `when "new", "queued"`. Το `when "new" || "queued"` θα υπολόγιζε πρώτα το `||` και θα έχανε τη δεύτερη περίπτωση.

Η παραλλαγή

Ομαδοποίησε κωδικούς 1 και 2 ως "small", με fallback "other". Δοκίμασε το 2.

```ruby
case 2
when 1, 2
  "small"
else
  "other"
end
```

Αποτέλεσμα: `"small"`.

009 / ΛΥΣΗ

Έξοδος νωρίς

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `0`.

```ruby
def seat_fee(n)
  return 0 unless n > 0
  n * 4
end
seat_fee(0)
```

Η κύρια άσκηση

```ruby
def booking_cost(seats, price, active)
  return 0 unless active
  return 0 if seats <= 0
  seats * price
end
```

Οι guards καλύπτουν τους κανόνες εξόδου. Αποφεύγουμε διπλές αρνήσεις όπως `unless !active`, γιατί δυσκολεύουν την ανάγνωση.

Η παραλλαγή

Δώσε "guest" μόνο για nil όνομα με guard. Για κενό String κράτησε την τιμή.

```ruby
def guest_name(name)
  return "guest" if name.nil?
  name
end
guest_name("")
```

Αποτέλεσμα: `""`.

010 / ΛΥΣΗ

Κανόνες αποστολής · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `0`.

```ruby
def pickup_fee(pickup)
  return 0 if pickup
  4
end
pickup_fee(true)
```

Η κύρια άσκηση

```ruby
def shipping_fee(weight, region, pickup)
  return 0 if pickup
  return nil if weight <= 0
  base = case region
  when "local"
    3
  when "remote"
    7
  else
    return nil
  end
  if weight > 5
    base + 2
  else
    base
  end
end
```

Το nil σημαίνει ότι δεν υπάρχει προσφορά, ενώ το 0 είναι έγκυρη δωρεάν παραλαβή. Αυτές οι δύο τιμές δεν πρέπει να συγχωνευτούν.

Η παραλλαγή

Σε χωριστή μέθοδο δώσε έκπτωση 2 σε ποσό τουλάχιστον 10 και κράτησε τα μικρότερα ποσά ίδια.

```ruby
def discounted_fee(amount)
  return amount if amount < 10
  amount - 2
end
discounted_fee(10)
```

Αποτέλεσμα: `8`.

011 / ΛΥΣΗ

Διατεταγμένες συλλογές

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "C", 3]`.

```ruby
stops = ["A", "B", "C"]
[stops[0], stops[-1], stops.length]
```

Η κύρια άσκηση

```ruby
def route_summary(stops)
  return "χωρίς στάσεις" if stops.empty?
  "#{stops.first} → #{stops.last} (#{stops.length})"
end
```

Η κενή λίστα χρειάζεται δικό της αποτέλεσμα πριν από την πρόσβαση. Η αρίθμηση θέσεων για τον άνθρωπο αρχίζει συνήθως από 1, οι δείκτες από 0.

Η παραλλαγή

Από τρεις στάσεις δώσε δεύτερη, τελευταία και ένα στοιχείο πέρα από το τέλος.

```ruby
stops = ["X", "Y", "Z"]
[stops[1], stops[-1], stops[3]]
```

Αποτέλεσμα: `["Y", "Z", nil]`.

012 / ΛΥΣΗ

Συγκέντρωση τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["tea", "water", "juice"]`.

```ruby
options = ["tea"]
options << "water"
options + ["juice"]
```

Η κύρια άσκηση

```ruby
def menu_options(coffee, dessert)
  options = ["νερό"]
  options << "καφές" if coffee
  options.push("γλυκό") if dessert
  options
end
```

Η τελευταία γραμμή είναι options. Αν τελείωνε σε modifier if με ψευδή συνθήκη, η επιστροφή θα ήταν nil.

Η παραλλαγή

Ένωσε αρχικές και έξτρα επιλογές και έλεγξε ότι το αρχικό Array παραμένει ίδιο.

```ruby
base = ["tea"]
combined = base + ["cake"]
[base, combined]
```

Αποτέλεσμα: `[["tea"], ["tea", "cake"]]`.

013 / ΛΥΣΗ

Ένα block για κάθε στοιχείο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `7`.

```ruby
total = 0
[2, 4, 1].each do |seats|
  total = total + seats
end
total
```

Η κύρια άσκηση

```ruby
def total_seats(bookings)
  total = 0
  bookings.each do |seats|
    total = total + seats
  end
  total
end
```

Ο συσσωρευτής ορίζεται πριν από το block και επιστρέφεται μετά. Αν η τελευταία γραμμή ήταν η each, η μέθοδος θα επέστρεφε το bookings.

Η παραλλαγή

Με each χτίσε νέο Array με το κείμενο "θέσεις=N" για κάθε κράτηση.

```ruby
labels = []
[1, 3].each { |n| labels << "θέσεις=#{n}" }
labels
```

Αποτέλεσμα: `["θέσεις=1", "θέσεις=3"]`.

014 / ΛΥΣΗ

Μία νέα τιμή για κάθε παλιά

Κάθε βαθμός δίνει μία ετικέτα. Το block υπολογίζει την ετικέτα και η `map` συγκεντρώνει τα αποτελέσματα.

```ruby
def grade_labels(scores)
  scores.map do |score|
    if score >= 90
      "άριστα"
    elsif score >= 50
      "πέρασε"
    else
      "κόπηκε"
    end
  end
end
```

Ελέγχουμε πρώτα το 90, επειδή ένας άριστος βαθμός περνά και το όριο 50. Το `if` είναι η τελευταία έκφραση του block και δίνει το String του επιλεγμένου κλάδου. Η κλήση `map` είναι η τελευταία έκφραση της μεθόδου, οπότε το νέο Array γίνεται και η επιστροφή της.

Στην κενή είσοδο το block δεν εκτελείται και η απάντηση είναι νέο κενό Array. Η λύση διαβάζει τους βαθμούς χωρίς να τους αλλάζει.

Ένα συνηθισμένο λάθος είναι να βάλεις `puts` ως τελευταία γραμμή του block. Τότε η τιμή που συγκεντρώνει η `map` είναι η επιστροφή της `puts`, δηλαδή `nil`. Η απάντηση μπορεί να φανεί στην οθόνη ενώ το επιστρεφόμενο Array είναι λάθος. Οι επιστροφές αυτές περιγράφονται στα Array#map και Kernel#puts.

Η παραλλαγή

Αποθηκεύουμε πρώτα την κατηγορία και επιστρέφουμε την αναλυτική ετικέτα ως τελευταία έκφραση.

```ruby
def detailed_grade_labels(scores)
  scores.map do |score|
    category = if score >= 90
      "άριστα"
    elsif score >= 50
      "πέρασε"
    else
      "κόπηκε"
    end

    "#{score}: #{category}"
  end
end
```

Η `detailed_grade_labels([49, 90])` επιστρέφει `["49: κόπηκε", "90: άριστα"]`.

Το solution.rb περιέχει την κύρια λύση. Η εντολή ελέγχου των λύσεων την ελέγχει χωρίς να αντικαταστήσει την προσπάθειά σου.

015 / ΛΥΣΗ

Κράτησε όσα ταιριάζουν

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[8, 8]`.

```ruby
[2, 8, 4, 8].select { |n| n >= 5 }
```

Η κύρια άσκηση

```ruby
def available_quantities(quantities, minimum, maximum)
  quantities.select do |quantity|
    quantity >= minimum && quantity <= maximum
  end
end
```

Το block απαντά ναι ή όχι για την αρχική ποσότητα. Αν έγραφες map, θα έπαιρνες booleans για όλες τις θέσεις.

Η παραλλαγή

Κράτησε ποσότητες που ισούνται ακριβώς με τον στόχο 4.

```ruby
[4, 2, 4, 5].select { |n| n == 4 }
```

Αποτέλεσμα: `[4, 4]`.

016 / ΛΥΣΗ

Αφαίρεσε όσα απορρίπτονται

Ένας βαθμός απορρίπτεται όταν βρίσκεται κάτω από το κάτω όριο ή πάνω από το πάνω όριο.

```ruby
def valid_scores(scores, minimum, maximum)
  scores.reject do |score|
    score < minimum || score > maximum
  end
end
```

Τα `<` και `>` αφήνουν τις τιμές που είναι ίσες με τα όρια. Το `||` συνδέει τους δύο ανεξάρτητους λόγους απόρριψης. Η `reject` κρατά όσες τιμές δεν ικανοποιούν αυτή τη συνθήκη.

Η μέθοδος επιστρέφει τη νέα λίστα που δίνει η `reject`. Δεν χρειάζεται ξεχωριστή μεταβλητή αποτελέσματος.

Ένα συνηθισμένο λάθος είναι να χρησιμοποιήσεις τη συνθήκη `score >= minimum && score <= maximum` μέσα στη `reject`. Αυτή περιγράφει τους έγκυρους βαθμούς, οπότε θα απορρίψεις ακριβώς αυτούς.

Η παραλλαγή

Για να κρατήσεις μόνο άκυρους βαθμούς με `reject`, απορρίπτεις τους έγκυρους.

```ruby
def invalid_scores(scores, minimum, maximum)
  scores.reject do |score|
    score >= minimum && score <= maximum
  end
end
```

Η `invalid_scores([-1, 0, 30, 100, 101], 0, 100)` επιστρέφει `[-1, 101]`.

Η σύγκριση των δύο λύσεων δείχνει γιατί διαβάζουμε πρώτα ποια συνθήκη γράφει το block. Η ίδια boolean τιμή έχει διαφορετικό ρόλο ανάλογα με το αν χρησιμοποιείται στη `select` ή στη `reject`. Δες τις αντίστοιχες υπογραφές στα Array#select και Array#reject.

Το solution.rb περιέχει την κύρια λύση. Η εντολή ελέγχου των λύσεων την ελέγχει χωρίς να αντικαταστήσει την προσπάθειά σου.

017 / ΛΥΣΗ

Μετατροπή ή φίλτρο; · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[41, 81], [80]]`.

```ruby
scores = [40, 80]
[scores.map { |n| n + 1 }, scores.select { |n| n >= 50 }]
```

Η κύρια άσκηση

```ruby
def split_scores(scores)
  labels = scores.map do |score|
    if score >= 0 && score <= 100
      "εντός"
    else
      "εκτός"
    end
  end
  valid = scores.select { |score| score >= 0 && score <= 100 }
  invalid = scores.reject { |score| score >= 0 && score <= 100 }
  [labels, valid, invalid]
end
```

Το πρώτο Array έχει πάντα το μέγεθος της εισόδου. Τα δύο φίλτρα μοιράζουν τα αρχικά στοιχεία σε συμπληρωματικές ομάδες.

Η παραλλαγή

Κράτησε πρώτα βαθμούς τουλάχιστον 50 και μετά πρόσθεσε 5 στον καθένα.

```ruby
[30, 50, 80].select { |n| n >= 50 }.map { |n| n + 5 }
```

Αποτέλεσμα: `[55, 85]`.

018 / ΛΥΣΗ

Το πρώτο που ταιριάζει

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `8`.

```ruby
[3, 8, 6].find { |n| n >= 5 }
```

Η κύρια άσκηση

```ruby
def first_stock(stocks, required)
  stocks.find { |stock| stock >= required }
end
```

Η find αρκεί για μία αντιστοίχιση. Το select θα συγκέντρωνε όλες σε Array και θα άλλαζε τη σύμβαση.

Η παραλλαγή

Μετέτρεψε απουσία αποτελέσματος σε "δεν βρέθηκε", κρατώντας την ποσότητα όταν υπάρχει.

```ruby
found = [1, 2].find { |n| n >= 4 }
if found.nil?
  "δεν βρέθηκε"
else
  found
end
```

Αποτέλεσμα: `"δεν βρέθηκε"`.

019 / ΛΥΣΗ

Ερωτήσεις για όλη τη συλλογή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[true, false]`.

```ruby
values = [0, 3]
[values.any? { |n| n > 0 }, values.all? { |n| n > 0 }]
```

Η κύρια άσκηση

```ruby
def passing_questions(scores)
  [
    scores.any? { |n| n >= 50 },
    scores.all? { |n| n >= 50 },
    scores.none? { |n| n >= 50 },
    scores.one? { |n| n >= 50 }
  ]
end
```

Το one? δεν σημαίνει «τουλάχιστον ένα». Η κενή all? δεν αποδεικνύει ότι η συλλογή έχει στοιχεία· για αυτό χρειάζεται και έλεγχος empty?.

Η παραλλαγή

Έλεγξε αν υπάρχει ακριβώς μία κενή λίστα μέσα σε δύο λίστες.

```ruby
[[1], []].one? { |items| items.empty? }
```

Αποτέλεσμα: `true`.

020 / ΛΥΣΗ

Πλήθος με κριτήριο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `2`.

```ruby
[2, 5, 2].count { |n| n == 2 }
```

Η κύρια άσκηση

```ruby
def fitting_bookings(bookings, capacity)
  bookings.count { |seats| seats > 0 && seats <= capacity }
end
```

Το block περιλαμβάνει και τον κανόνα seats > 0. Η απλή σύγκριση <= capacity θα μετρούσε κενές κρατήσεις.

Η παραλλαγή

Μέτρησε τις κρατήσεις που υπερβαίνουν χωρητικότητα 3.

```ruby
[1, 4, 3, 5].count { |n| n > 3 }
```

Αποτέλεσμα: `2`.

021 / ΛΥΣΗ

Άθροισμα με μετατροπή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `20`.

```ruby
[2, 3].sum { |n| n * 4 }
```

Η κύρια άσκηση

```ruby
def booking_revenue(bookings, price, fee)
  bookings.sum { |seats| seats * price + fee }
end
```

Το fee μπαίνει μέσα στο block γιατί ισχύει ανά κράτηση. Πάγιο για όλη τη λίστα θα έμπαινε ως αρχική τιμή ή έξω από τη sum.

Η παραλλαγή

Άρχισε από πάγιο 10 και πρόσθεσε τις δοσμένες χρεώσεις 2 και 3.

```ruby
[2, 3].sum(10)
```

Αποτέλεσμα: `15`.

022 / ΛΥΣΗ

Θέση μαζί με τιμή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["1. A", "2. B"]`.

```ruby
rows = []
["A", "B"].each_with_index { |name, index| rows << "#{index + 1}. #{name}" }
rows
```

Η κύρια άσκηση

```ruby
def ranking_lines(names)
  lines = []
  names.each_with_index do |name, index|
    lines << "#{index + 1}. #{name}"
  end
  lines
end
```

Η σειρά των παραμέτρων είναι τιμή και μετά δείκτης. Το index αλλάζει για κάθε θέση, ακόμη κι αν το όνομα επαναλαμβάνεται.

Η παραλλαγή

Φτιάξε αρίθμηση από το 10 με την ίδια each_with_index.

```ruby
rows = []
["A", "B"].each_with_index { |name, i| rows << "#{i + 10}: #{name}" }
rows
```

Αποτέλεσμα: `["10: A", "11: B"]`.

023 / ΛΥΣΗ

Μικρό βαθμολόγιο · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[40, 80], 1]`.

```ruby
valid = [-1, 40, 80].select { |n| n >= 0 }
[valid, valid.count { |n| n >= 50 }]
```

Η κύρια άσκηση

```ruby
def gradebook(scores)
  valid = scores.select { |n| n >= 0 && n <= 100 }
  labels = valid.map do |n|
    if n >= 50
      "πέρασε"
    else
      "κόπηκε"
    end
  end
  [valid, labels, valid.count { |n| n >= 50 }]
end
```

Η μέτρηση χρησιμοποιεί valid, όχι scores. Αλλιώς ο άκυρος βαθμός 101 θα μετρούσε ως επιτυχία.

Η παραλλαγή

Στους έγκυρους βαθμούς [0, 50, 100] μέτρησε πόσοι είναι κάτω από 50.

```ruby
[0, 50, 100].count { |n| n < 50 }
```

Αποτέλεσμα: `1`.

024 / ΛΥΣΗ

Καθάρισμα άκρων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"A B"`.

```ruby
"  A  B\n".strip
```

Η κύρια άσκηση

```ruby
def clean_names(names)
  names.map { |name| name.strip }
end
```

Η strip εφαρμόζεται σε κάθε όνομα μέσα στη map. Η chomp διατηρεί τα κενά γύρω από το όνομα και δεν την υποκαθιστά.

Η παραλλαγή

Αφαίρεσε μόνο το newline από " A \n".

```ruby
" A \n".chomp
```

Αποτέλεσμα: `" A "`.

025 / ΛΥΣΗ

Πεζά και κεφαλαία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Ruby code"`.

```ruby
"rUBY CODE".capitalize
```

Η κύρια άσκηση

```ruby
def code_labels(codes)
  codes.map { |code| code.strip.upcase }
end
```

Η σειρά strip και μετά upcase είναι εύκολη στην ανάγνωση. Η capitalize θα έδινε Ab, όχι AB.

Η παραλλαγή

Φτιάξε πεζούς κωδικούς από [" AB ", "Cd"].

```ruby
[" AB ", "Cd"].map { |s| s.strip.downcase }
```

Αποτέλεσμα: `["ab", "cd"]`.

026 / ΛΥΣΗ

Κείμενο σε πεδία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "2"]`.

```ruby
"A:2".split(":")
```

Η κύρια άσκηση

```ruby
def parse_quantities(lines)
  lines.map do |line|
    fields = line.split(":")
    [fields[0], fields[1].to_i]
  end
end
```

Η split επιστρέφει Strings και το δεύτερο πεδίο θέλει to_i. Η σύμβαση εγγυάται έγκυρη μορφή, οπότε εδώ δεν γράφουμε parser σφαλμάτων.

Η παραλλαγή

Χώρισε το "A:" κρατώντας το τελικό κενό πεδίο.

```ruby
"A:".split(":", -1)
```

Αποτέλεσμα: `["A", ""]`.

027 / ΛΥΣΗ

Πεδία σε κείμενο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"tea / cake"`.

```ruby
["tea", "cake"].join(" / ")
```

Η κύρια άσκηση

```ruby
def text_report(rows)
  rows.map { |row| row.join(" | ") }.join("\n")
end
```

Η εσωτερική join ενώνει πεδία, η εξωτερική γραμμές. Η προσθήκη newline μέσα στη map θα έβαζε και τελικό newline.

Η παραλλαγή

Ένωσε ονόματα με κόμμα και κενό.

```ruby
["Eva", "Jo"].join(", ")
```

Αποτέλεσμα: `"Eva, Jo"`.

028 / ΛΥΣΗ

Διαστήματα τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[2, 3, 4], [2, 3]]`.

```ruby
[(2..4).to_a, (2...4).to_a]
```

Η κύρια άσκηση

```ruby
def discount_band(quantity)
  if (0..4).cover?(quantity)
    "small"
  elsif (5..9).cover?(quantity)
    "medium"
  else
    "large"
  end
end
```

Τα διαστήματα δεν αφήνουν κενά. Το ... αποκλείει μόνο το δεξιό άκρο, όχι και το αριστερό.

Η παραλλαγή

Παρήγαγε τις θέσεις 3 έως 6, χωρίς το 6.

```ruby
(3...6).to_a
```

Αποτέλεσμα: `[3, 4, 5]`.

029 / ΛΥΣΗ

Κομμάτια από μια τιμή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["B", "C"], "BC"]`.

```ruby
[["A", "B", "C"][1, 2], "ABCDE"[1...3]]
```

Η κύρια άσκηση

```ruby
def result_page(items, offset, size)
  items[offset, size] || []
end
```

Το δεύτερο όρισμα είναι πλήθος, όχι τελικός δείκτης. Το `|| []` καλύπτει το `nil` της πρόσβασης εκτός ορίων.

Η παραλλαγή

Πάρε τους τρεις πρώτους χαρακτήρες του `"RUBY-42"`.

```ruby
"RUBY-42"[0, 3]
```

Αποτέλεσμα: `"RUB"`.

030 / ΛΥΣΗ

Απλή αναζήτηση κειμένου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[true, false]`.

```ruby
["report.csv".end_with?(".csv"), "x-report.csv".start_with?("report")]
```

Η κύρια άσκηση

```ruby
def matching_files(names, prefix, suffix)
  names.select { |name| name.start_with?(prefix) && name.end_with?(suffix) }
end
```

Η include? δεν επιβάλλει θέση. Για το prefix χρειαζόμαστε start_with?.

Η παραλλαγή

Κράτησε ονόματα που περιέχουν "draft" οπουδήποτε.

```ruby
["my-draft.txt", "final.txt"].select { |s| s.include?("draft") }
```

Αποτέλεσμα: `["my-draft.txt"]`.

031 / ΛΥΣΗ

Αντικατάσταση σταθερού κειμένου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A-X", "A-A"]`.

```ruby
["X-X".sub("X", "A"), "X-X".gsub("X", "A")]
```

Η κύρια άσκηση

```ruby
def fill_template(text, name)
  text.gsub("{name}", name)
end
```

Η gsub καλύπτει επαναλαμβανόμενα tokens. Με sub το δεύτερο {name} θα έμενε αμετάβλητο.

Η παραλλαγή

Αντικατάστησε μόνο το πρώτο "TODO" με "DONE".

```ruby
"TODO, TODO".sub("TODO", "DONE")
```

Αποτέλεσμα: `"DONE, TODO"`.

032 / ΛΥΣΗ

Χαρακτήρες και γραμμές

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "B"]`.

```ruby
"A\nB\n".lines.map { |line| line.chomp }
```

Η κύρια άσκηση

```ruby
def numbered_text(text)
  result = []
  text.lines.each_with_index do |line, index|
    result << "#{index + 1}: #{line.chomp}"
  end
  result
end
```

Η chomp αφήνει τα χρήσιμα κενά της γραμμής. Η strip θα τα αφαιρούσε. Το τελικό newline δεν δημιουργεί πρόσθετη κενή γραμμή στη lines.

Η παραλλαγή

Με each_char μέτρησε πόσες φορές υπάρχει ο χαρακτήρας a στο "aba".

```ruby
count = 0
"aba".each_char { |char| count = count + 1 if char == "a" }
count
```

Αποτέλεσμα: `2`.

033 / ΛΥΣΗ

Καθάρισμα λίστας ονομάτων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A"]`.

```ruby
[" A ", " "].map { |s| s.strip }.reject { |s| s.empty? }
```

Η κύρια άσκηση

```ruby
def name_lines(names)
  names.map { |s| s.strip.capitalize }.reject { |s| s.empty? }.join("\n")
end
```

Αν απορρίψεις empty? πριν τη strip, ένα String γεμάτο κενά θα περάσει το φίλτρο.

Η παραλλαγή

Με τα ίδια βήματα φτιάξε μία γραμμή με διαχωριστή ", ".

```ruby
[" eVA ", " ", "JO"].map { |s| s.strip.capitalize }.reject { |s| s.empty? }.join(",
")
```

Αποτέλεσμα: `"Eva, Jo"`.

034 / ΛΥΣΗ

Μεταβολή του ίδιου αντικειμένου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["abc", nil]`.

```ruby
text = "abc"
returned = text.strip!
[text, returned]
```

Η κύρια άσκηση

```ruby
def normalize_in_place(text)
  text.strip!
  text.upcase!
  text
end
```

Καλούμε τις bang μεθόδους σε χωριστές γραμμές και επιστρέφουμε text. Η αλυσίδα strip!.upcase! θα μπορούσε να καλέσει upcase! πάνω σε nil.

Η παραλλαγή

Παρατήρησε τι επιστρέφει η upcase! όταν το κείμενο είναι ήδη κεφαλαίο.

```ruby
text = "OK"
[text.upcase!, text]
```

Αποτέλεσμα: `[nil, "OK"]`.

035 / ΛΥΣΗ

Δύο ονόματα, ένα αντικείμενο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["A!"], ["A!"]]`.

```ruby
original = ["A"]
copy = original.dup
copy[0] << "!"
[original, copy]
```

Η κύρια άσκηση

```ruby
def loud_title(title)
  copy = title.dup
  copy.upcase!
  copy
end
```

Η dup αντιγράφει το String σε αυτό το μάθημα. Για δομή πολλών επιπέδων χρειάζεται να αποφασίσεις ποια εσωτερικά αντικείμενα θα είναι ανεξάρτητα.

Η παραλλαγή

Αντίγραψε και κάθε εσωτερικό String ώστε η προσθήκη ! στο αντίγραφο να μη μεταβάλλει το αρχικό Array.

```ruby
original = ["A"]
copy = original.map { |s| s.dup }
copy[0] << "!"
[original, copy]
```

Αποτέλεσμα: `[["A"], ["A!"]]`.

036 / ΛΥΣΗ

Καθαρές ετικέτες συμμετοχής · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["[JO]"]`.

```ruby
[" jo "].map { |s| "[#{s.strip.upcase}]" }
```

Η κύρια άσκηση

```ruby
def participant_badges(names)
  names.map { |s| s.strip.upcase }.reject { |s| s.empty? }.map { |s| "[#{s}]" }
end
```

Το φίλτρο προηγείται της προσθήκης αγκυλών. Το "[]" δεν είναι κενό String, οπότε φίλτρο στο τέλος θα κρατούσε άδειες ετικέτες.

Η παραλλαγή

Φτιάξε πεζές ετικέτες με πρόθεμα @, για τα ίδια είδη εισόδου.

```ruby
[" EVA ", " "].map { |s| s.strip.downcase }.reject { |s| s.empty? }.map { |s|
"@#{s}" }
```

Αποτέλεσμα: `["@eva"]`.

037 / ΛΥΣΗ

Εγγραφές με ονομασμένα πεδία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva", 3, [:name, :seats]]`.

```ruby
record = {name: "Eva", seats: 2}
record[:seats] = 3
[record[:name], record[:seats], %i[name seats]]
```

Η κύρια άσκηση

```ruby
def booking_record(name, seats)
  record = {name: name, seats: seats, status: "pending"}
  [record, "#{record[:name]}: #{record[:seats]} (#{record[:status]})"]
end
```

Το Hash δίνει ονόματα στα πεδία. Η πρόσβαση με "name" θα επέστρεφε nil, επειδή το κλειδί είναι :name.

Η παραλλαγή

Άλλαξε μόνο την κατάσταση μιας νέας εγγραφής σε "ready".

```ruby
record = {name: "Jo", status: "pending"}
record[:status] = "ready"
record
```

Αποτέλεσμα: `{name: "Jo", status: "ready"}`.

038 / ΛΥΣΗ

Λείπει κλειδί ή υπάρχει τιμή nil;

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[false, nil, 4]`.

```ruby
settings = {enabled: false, title: nil}
[settings.fetch(:enabled, true), settings.fetch(:title, "X"), settings.fetch(:size, 4)]
```

Η κύρια άσκηση

```ruby
def effective_settings(settings)
  {enabled: settings.fetch(:enabled, true), title: settings.fetch(:title, "Guest"), title_given: settings.key?(:title)}
end
```

Το settings[:enabled] || true θα έσβηνε την επιλογή false. Η fetch διατηρεί όλες τις αποθηκευμένες τιμές.

Η παραλλαγή

Διάβασε limit με προεπιλογή 10 από Hash που περιέχει limit: 0.

```ruby
{limit: 0}.fetch(:limit, 10)
```

Αποτέλεσμα: `0`.

039 / ΛΥΣΗ

Κλειδί και τιμή στο block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[:tea, :cake], [4, nil]]`.

```ruby
prices = {tea: 2, cake: 4}
[prices.keys, prices.values_at(:cake, :water)]
```

Η κύρια άσκηση

```ruby
def price_lines(prices)
  lines = []
  prices.each { |code, price| lines << "#{code}: #{price}" }
  lines
end
```

Οι παράμετροι είναι code και price, όχι δείκτης και στοιχείο όπως σε αριθμημένη λίστα.

Η παραλλαγή

Πάρε τις τιμές με σειρά tea, cake, ακόμη κι αν τα κλειδιά μπήκαν ανάποδα.

```ruby
{cake: 4, tea: 2}.values_at(:tea, :cake)
```

Αποτέλεσμα: `[2, 4]`.

040 / ΛΥΣΗ

Δεδομένα μέσα σε δεδομένα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Athens"`.

```ruby
{orders: [{city: "Athens"}]}.dig(:orders, 0, :city)
```

Η κύρια άσκηση

```ruby
def shipping_city(order)
  city = order.dig(:shipping, :address, :city)
  return "unknown" if city.nil?
  city
end
```

Η dig καλύπτει την απουσία στις συμφωνημένες δομές. Η σύμβαση δεν ζητά χειρισμό αυθαίρετων τύπων.

Η παραλλαγή

Διάβασε την πρώτη ετικέτα από εγγραφή με κενό Array labels.

```ruby
{labels: []}.dig(:labels, 0)
```

Αποτέλεσμα: `nil`.

041 / ΛΥΣΗ

Συγχώνευση ρυθμίσεων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{color: "blue", active: false}`.

```ruby
{color: "blue", active: true}.merge({active: false})
```

Η κύρια άσκηση

```ruby
def apply_options(defaults, options)
  defaults.merge(options)
end
```

Η σειρά defaults.merge(options) υλοποιεί προτεραιότητα χρήστη. Αν την αντιστρέψεις, τα defaults θα σβήσουν επιλογές.

Η παραλλαγή

Πρόβλεψε συγχώνευση δύο nested ρυθμίσεων.

```ruby
{view: {color: "blue", size: 2}}.merge({view: {size: 3}})
```

Αποτέλεσμα: `{view: {size: 3}}`.

042 / ΛΥΣΗ

Προεπιλογές εγγραφής · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `false`.

```ruby
{enabled: true}.merge({enabled: false}).fetch(:enabled)
```

Η κύρια άσκηση

```ruby
def booking_description(record, defaults)
  settings = defaults.merge(record)
  state = if settings.fetch(:active)
    "active"
  else
    "inactive"
  end
  "#{settings.fetch(:name)}: #{state}"
end
```

Δεν χρησιμοποιούμε || για active. Η merge κρατά το false που έδωσε η εγγραφή.

Η παραλλαγή

Διάβασε όνομα Guest όταν λείπει, αλλά κράτησε ρητό κενό String.

```ruby
{name: ""}.fetch(:name, "Guest")
```

Αποτέλεσμα: `""`.

043 / ΛΥΣΗ

Μετατροπή κλειδιών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"a" => 2, "b" => 4}`.

```ruby
{"A" => 2, "B" => 4}.transform_keys { |key| key.downcase }
```

Η κύρια άσκηση

```ruby
def normalized_keys(record)
  record.transform_keys { |key| key.strip.downcase }
end
```

Το block παίρνει κλειδί, όχι κλειδί και τιμή. Η μετατροπή μπορεί να μειώσει το πλήθος κλειδιών λόγω σύγκρουσης.

Η παραλλαγή

Κάνε μόνο κεφαλαία τα κλειδιά μιας μικρής εγγραφής.

```ruby
{"x" => 0}.transform_keys { |key| key.upcase }
```

Αποτέλεσμα: `{"X" => 0}`.

044 / ΛΥΣΗ

Μετατροπή τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{tea: 3, cake: 5}`.

```ruby
{tea: 2, cake: 4}.transform_values { |price| price + 1 }
```

Η κύρια άσκηση

```ruby
def adjust_prices(prices, increase)
  prices.transform_values { |price| price + increase }
end
```

Η map πάνω σε Hash θα έδινε Array. Η transform_values κρατά το Hash ως σχήμα εξόδου.

Η παραλλαγή

Διπλασίασε ποσότητες στο Hash {a: 2, b: 0}.

```ruby
{a: 2, b: 0}.transform_values { |n| n * 2 }
```

Αποτέλεσμα: `{a: 4, b: 0}`.

045 / ΛΥΣΗ

Επιλογή πεδίων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{name: "Eva", seats: 2}`.

```ruby
{name: "Eva", seats: 2, note: "x"}.slice(:name, :seats)
```

Η κύρια άσκηση

```ruby
def public_record(record)
  record.slice(:name, :seats)
end
```

Η slice επιλέγει ρητά δημόσια πεδία. Η delete θα μετέβαλλε την είσοδο, εκτός αν δούλευες σε αντίγραφο.

Η παραλλαγή

Σε αντίγραφο εγγραφής αφαίρεσε note με delete και δώσε [αφαιρεμένη τιμή, αντίγραφο, αρχικό].

```ruby
original = {name: "A", note: "x"}
copy = original.dup
removed = copy.delete(:note)
[removed, copy, original]
```

Αποτέλεσμα: `["x", {name: "A"}, {name: "A", note: "x"}]`.

046 / ΛΥΣΗ

Λίστα παραγγελιών · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{id: 2, ready: true}`.

```ruby
[{id: 1, ready: false}, {id: 2, ready: true}].find { |r| r[:ready] }
```

Η κύρια άσκηση

```ruby
def order_overview(orders)
  active = orders.reject { |order| order[:cancelled] }
  {first_ready: active.find { |order| order[:ready] }, labels: active.map { |order| "Order #{order[:id]}" }}
end
```

Τα μεταγενέστερα βήματα χρησιμοποιούν active. Η ακυρωμένη εγγραφή δεν πρέπει να επανεμφανιστεί μέσω του find.

Η παραλλαγή

Κράτησε μόνο ready εγγραφές και δώσε τα ids τους.

```ruby
[{id: 1, ready: false}, {id: 2, ready: true}].select { |r| r[:ready] }.map { |r| r[:id] }
```

Αποτέλεσμα: `[2]`.

047 / ΛΥΣΗ

Προαιρετικά ορίσματα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hi Eva"`.

```ruby
def greeting(name, prefix = "Hi")
  "#{prefix} #{name}"
end
greeting("Eva")
```

Η κύρια άσκηση

```ruby
def item_label(code, prefix = "ID")
  "#{prefix}:#{code}"
end
```

Το default ανήκει στην υπογραφή. Δεν χρειάζεται να μαντεύεις μέσα στο σώμα αν ο χρήστης έδωσε κενό String.

Η παραλλαγή

Γράψε χαιρετισμό με default "Hello" και ρητό prefix "Hey".

```ruby
def welcome(name, prefix = "Hello")
  "#{prefix} #{name}"
end
welcome("Jo", "Hey")
```

Αποτέλεσμα: `"Hey Jo"`.

048 / ΛΥΣΗ

Ονομασμένα ορίσματα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `9`.

```ruby
def delivery(weight:, remote: false)
  base = if remote
    7
  else
    3
  end
  weight + base
end
delivery(remote: true, weight: 2)
```

Η κύρια άσκηση

```ruby
def delivery_quote(weight:, remote: false, pickup: false)
  return 0 if pickup
  base = if remote
    7
  else
    3
  end
  base + weight
end
```

Τα keywords κάνουν σαφές ποια επιλογή αλλάζει. Το required weight: δεν έχει default και πρέπει να δοθεί από τον καλούντα.

Η παραλλαγή

Φτιάξε formatter με υποχρεωτικό name: και προαιρετικό suffix: "!".

```ruby
def keyword_greeting(name:, suffix: "!")
  "#{name}#{suffix}"
end
keyword_greeting(name: "Eva", suffix: "?")
```

Αποτέλεσμα: `"Eva?"`.

049 / ΛΥΣΗ

Όρια ορατότητας

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"outside"`.

```ruby
label = "outside"
[1].each do |n; label|
  label = "inside #{n}"
end
label
```

Η κύρια άσκηση

```ruby
def scoped_labels(numbers)
  label = "original"
  labels = []
  numbers.each do |number; label|
    label = "#{number}!"
    labels << label
  end
  [label, labels]
end
```

Το semicolon χωρίζει παραμέτρους από block-local ονόματα. Χωρίς αυτό, η ανάθεση label θα άλλαζε το εξωτερικό label.

Η παραλλαγή

Με block άλλαξε ρητά εξωτερικό total από 0 σε άθροισμα 2 και 3.

```ruby
total = 0
[2, 3].each { |n| total = total + n }
total
```

Αποτέλεσμα: `5`.

050 / ΛΥΣΗ

Προαιρετικός receiver

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `nil`.

```ruby
name = nil
name&.strip&.upcase
```

Η κύρια άσκηση

```ruby
def optional_badge(name)
  name&.strip&.upcase
end
```

Το name&.strip.upcase δεν προστατεύει τη δεύτερη κλήση όταν το πρώτο αποτέλεσμα είναι nil.

Η παραλλαγή

Μετά την ασφαλή μετατροπή, δώσε fallback "Guest" για nil.

```ruby
name = nil
name&.upcase || "Guest"
```

Αποτέλεσμα: `"Guest"`.

051 / ΛΥΣΗ

Ανάθεση υπό συνθήκη

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `true`.

```ruby
flag = false
flag ||= true
flag
```

Η κύρια άσκηση

```ruby
def fill_label(settings)
  settings[:label] ||= "Guest"
  settings
end
```

Το ||= ταιριάζει επειδή η εκφώνηση θεωρεί false απουσία label. Δεν είναι σωστή γενική επιλογή για boolean ρυθμίσεις.

Η παραλλαγή

Με &&= κάνε κεφαλαίο ένα προαιρετικό String όταν υπάρχει.

```ruby
label = "guest"
label &&= label.upcase
label
```

Αποτέλεσμα: `"GUEST"`.

052 / ΛΥΣΗ

Μικρές διορθώσεις syntax · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[false, nil]`.

```ruby
record = {active: false, name: nil}
[record.fetch(:active, true), record[:name]&.upcase]
```

Η κύρια άσκηση

```ruby
def safe_summary(record)
  {active: record.fetch(:active, true), name: record[:name]&.strip&.upcase ||
"Guest"}
end
```

Η fetch διακρίνει απόν κλειδί. Η &. χειρίζεται nil receiver. Ο κάθε μηχανισμός καλύπτει διαφορετικό μέρος της σύμβασης.

Η παραλλαγή

Διατήρησε ρητό nil σε enabled με fetch και default true.

```ruby
{enabled: nil}.fetch(:enabled, true)
```

Αποτέλεσμα: `nil`.

053 / ΛΥΣΗ

Ταξινόμηση έτοιμων τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["B2", "B1", "A1"]`.

```ruby
["B2", "A1", "B1"].sort.reverse
```

Η κύρια άσκηση

```ruby
def code_orders(codes)
  ascending = codes.sort
  [ascending, ascending.reverse]
end
```

Η reverse από μόνη της δεν ταξινομεί. Πρώτα sort και μετά reverse δίνουν φθίνουσα σειρά.

Η παραλλαγή

Ταξινόμησε τους ακεραίους 10, 2, 1 και σύγκρινε με την ίδια λίστα ως Strings.

```ruby
[[10, 2, 1].sort, ["10", "2", "1"].sort]
```

Αποτέλεσμα: `[[1, 2, 10], ["1", "10", "2"]]`.

054 / ΛΥΣΗ

Ταξινόμηση με κλειδί

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["a", "cc", "bbbb"]`.

```ruby
["bbbb", "a", "cc"].sort_by { |s| s.length }
```

Η κύρια άσκηση

```ruby
def ordered_tasks(tasks)
  tasks.sort_by { |task| [task[:priority], task[:name]] }
end
```

Το block επιστρέφει κλειδί, όχι boolean. Η Ruby αναλαμβάνει τον αλγόριθμο ταξινόμησης.

Η παραλλαγή

Ταξινόμησε ονόματα πρώτα κατά μήκος και μετά αλφαβητικά.

```ruby
["Jo", "Al", "Eva"].sort_by { |name| [name.length, name] }
```

Αποτέλεσμα: `["Al", "Jo", "Eva"]`.

055 / ΛΥΣΗ

Μικρότερο και μεγαλύτερο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{name: "B", price: 2}`.

```ruby
[{name: "A", price: 5}, {name: "B", price: 2}].min_by { |p| p[:price] }
```

Η κύρια άσκηση

```ruby
def cheapest_product(products)
  products.select { |p| p[:available] }.min_by { |p| [p[:price], p[:code]] }
end
```

Η min_by επιστρέφει προϊόν, όχι το price που παρήγαγε το block. Αυτό διατηρεί τα υπόλοιπα πεδία διαθέσιμα.

Η παραλλαγή

Δώσε μόνο μικρότερη και μεγαλύτερη αριθμητική τιμή από [8, 2, 5].

```ruby
prices = [8, 2, 5]
[prices.min, prices.max]
```

Αποτέλεσμα: `[2, 8]`.

056 / ΛΥΣΗ

Μία εμφάνιση ανά ταυτότητα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "B"]`.

```ruby
["A", "B", "A"].uniq
```

Η κύρια άσκηση

```ruby
def first_products(products)
  products.uniq { |p| p[:code] }
end
```

Αν κάνεις map του code και μετά uniq, θα χάσεις τις εγγραφές. Το block της uniq δίνει μόνο κλειδί ταυτότητας.

Η παραλλαγή

Κράτησε ένα όνομα ανά πεζή μορφή χωρίς να αλλάξεις τη γραφή της πρώτης εμφάνισης.

```ruby
["Eva", "EVA", "Jo"].uniq { |s| s.downcase }
```

Αποτέλεσμα: `["Eva", "Jo"]`.

057 / ΛΥΣΗ

Αφαίρεση απουσίας

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[false, 0, ""]`.

```ruby
[nil, false, 0, ""].compact
```

Η κύρια άσκηση

```ruby
def present_values(values)
  values.compact
end
```

Η reject { |value| !value } θα αφαιρούσε και false. Η compact εκφράζει ακριβώς την απουσία nil.

Η παραλλαγή

Καθάρισε προαιρετικά ονόματα από nil και ένωσε όσα έμειναν.

```ruby
["Eva", nil, "Jo"].compact.join(", ")
```

Αποτέλεσμα: `"Eva, Jo"`.

058 / ΛΥΣΗ

Ομάδες με κοινό γνώρισμα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{1 => ["a", "c"], 2 => ["bb"]}`.

```ruby
["a", "bb", "c"].group_by { |s| s.length }
```

Η κύρια άσκηση

```ruby
def products_by_category(products)
  products.group_by { |product| product[:category] }
end
```

Το block επιστρέφει κατηγορία. Οι τιμές του τελικού Hash είναι Arrays, ακόμη και για ομάδα με ένα μόνο προϊόν.

Η παραλλαγή

Ομαδοποίησε ποσότητες σε true/false ανάλογα αν είναι τουλάχιστον 5.

```ruby
[2, 5, 8].group_by { |n| n >= 5 }
```

Αποτέλεσμα: `{false => [2], true => [5, 8]}`.

059 / ΛΥΣΗ

Συχνότητες τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"tea" => 2, "cake" => 1}`.

```ruby
["tea", "cake", "tea"].tally
```

Η κύρια άσκηση

```ruby
def choice_counts(choices)
  choices.map { |choice| choice.strip.downcase }.tally
end
```

Η tally δίνει μετρητές, ενώ η group_by δίνει ολόκληρα Arrays στοιχείων. Διάλεξε σύμφωνα με την έξοδο που χρειάζεσαι.

Η παραλλαγή

Μέτρησε έτοιμες boolean επιλογές, κρατώντας και false.

```ruby
[true, false, true].tally
```

Αποτέλεσμα: `{true => 2, false => 1}`.

060 / ΛΥΣΗ

Μικρές ομάδες προϊόντων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"x" => [{code: "A", group: "x"}]}`.

```ruby
[{code: "A", group: "x"}, {code: "A", group: "y"}].uniq { |p| p[:code] }.group_by { |p| p[:group] }
```

Η κύρια άσκηση

```ruby
def category_counts(products)
  products.uniq { |p| p[:code] }.group_by { |p| p[:category] }.transform_values { |group| group.length }
end
```

Το block της transform_values παίρνει πλέον ολόκληρη ομάδα. Το στάδιο των δεδομένων καθορίζει τι είναι η παράμετρος.

Η παραλλαγή

Αντί για πλήθη, δώσε τους κωδικούς κάθε ομάδας.

```ruby
[{code: "A", category: "x"}, {code: "B", category: "x"}].group_by { |p| p[:category] }.transform_values { |group| group.map { |p| p[:code] } }
```

Αποτέλεσμα: `{"x" => ["A", "B"]}`.

061 / ΛΥΣΗ

Δύο συμπληρωματικές ομάδες

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[7], [2, 1]]`.

```ruby
[2, 7, 1].partition { |n| n >= 5 }
```

Η κύρια άσκηση

```ruby
def readiness_groups(records)
  records.partition { |record| record[:ready] }
end
```

Η πρώτη ομάδα αντιστοιχεί στην αλήθεια του block. Αν γράψεις !record[:ready], αντιστρέφεις τη σειρά εξόδου.

Η παραλλαγή

Χώρισε ονόματα σε κενά και μη κενά.

```ruby
["A", "", "B"].partition { |s| s.empty? }
```

Αποτέλεσμα: `[[""], ["A", "B"]]`.

062 / ΛΥΣΗ

Μετατροπή μαζί με απόρριψη

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["N3", "N2"]`.

```ruby
[1, 3, 2].filter_map { |n| "N#{n}" if n >= 2 }
```

Η κύρια άσκηση

```ruby
def ready_labels(records)
  records.filter_map { |record| "Ready #{record[:id]}" if record[:ready] }
end
```

Αν πρέπει να διατηρηθεί false ως αποτέλεσμα, η filter_map δεν ταιριάζει. Η map μαζί με compact αφαιρεί μόνο nil.

Η παραλλαγή

Με filter_map επίστρεψε την ίδια τιμή από [nil, false, 0, ""]. Πρόβλεψε τι μένει.

```ruby
[nil, false, 0, ""].filter_map { |value| value }
```

Αποτέλεσμα: `[0, ""]`.

063 / ΛΥΣΗ

Πολλά αποτελέσματα από κάθε στοιχείο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[1, 10, 2, 20]`.

```ruby
[1, 2].flat_map { |n| [n, n * 10] }
```

Η κύρια άσκηση

```ruby
def all_tags(products)
  products.flat_map { |product| product[:tags] }
end
```

Η map θα επέστρεφε Array από Arrays. Η flat_map ενώνει ακριβώς το ένα επίπεδο που δημιουργεί το block.

Η παραλλαγή

Δώσε δύο labels για κάθε όνομα: κανονικό και κεφαλαίο.

```ruby
["Eva", "Jo"].flat_map { |name| [name, name.upcase] }
```

Αποτέλεσμα: `["Eva", "EVA", "Jo", "JO"]`.

064 / ΛΥΣΗ

Παρατήρηση μέσα σε αλυσίδα

Το νέο βήμα μπαίνει εκεί όπου υπάρχουν τα καθαρά μη κενά ονόματα, πριν αλλάξει η σειρά τους.

```ruby
def name_report(names, audit)
  names
    .map { |name| name.strip }
    .reject { |name| name.empty? }
    .tap { |cleaned_names| audit << cleaned_names }
    .sort
    .join(", ")
end
```

Η `tap` δίνει ολόκληρη την ενδιάμεση λίστα στο block. Η `audit << cleaned_names` την προσθέτει ως μία εγγραφή στο audit.

Η προσθήκη με `<<` επιστρέφει το ίδιο το audit. Η `tap` αγνοεί αυτή την τιμή επιστροφής του block και επιστρέφει τη λίστα `cleaned_names`. Έτσι η επόμενη `sort` ταξινομεί τα ονόματα.

Η `sort` δίνει νέο Array. Η καταγεγραμμένη λίστα κρατά λοιπόν τη σειρά πριν από την ταξινόμηση. Δεν χρειάζεται επιπλέον αντίγραφο γι' αυτή τη συγκεκριμένη αλυσίδα. Αν το επόμενο βήμα άλλαζε τον ίδιο receiver, θα έπρεπε να ξανασκεφτούμε τη σύμβαση της καταγραφής.

Η `tap` δεν σημαίνει «το αντικείμενο δεν αλλάζει». Σημαίνει «επιστρέφεται ο ίδιος receiver αφού εκτελεστεί το block». Οι σχετικές συμβάσεις βρίσκονται στα Kernel#tap, Array#<< και Array#sort.

Στην κενή είσοδο η ενδιάμεση λίστα είναι `[]`. Η `tap` εκτελείται κανονικά μία φορά, οπότε προσθέτει αυτή την κενή λίστα στο audit.

Η παραλλαγή

Καταγράφουμε πλήθος πριν από τις μετατροπές και πλήθος μετά το φίλτρο.

```ruby
def name_report_with_counts(names, audit)
  names
    .tap { |items| audit << items.length }
    .map { |name| name.strip }
    .reject { |name| name.empty? }
    .tap { |items| audit << items.length }
    .sort
    .join(", ")
end
```

Για `[" Nikos ", " ", "Anna"]` η έξοδος είναι `"Anna, Nikos"` και στο αρχικά κενό audit προστίθενται οι τιμές `[3, 2]`.

Το επόμενο μάθημα του χάρτη, το 065, εισάγει τη `then`, που δίνει διαφορετικό ρόλο στην τιμή επιστροφής του block.

Το solution.rb περιέχει την κύρια λύση. Η εντολή ελέγχου των λύσεων την ελέγχει χωρίς να αντικαταστήσει την προσπάθειά σου.

065 / ΛΥΣΗ

Συνέχεια με το αποτέλεσμα του block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Total: 5"`.

```ruby
[2, 3].then { |numbers| "Total: #{numbers.sum}" }
```

Η κύρια άσκηση

```ruby
def names_summary(names)
  names.map { |s| s.strip }.reject { |s| s.empty? }.then do |cleaned|
    {count: cleaned.length, text: cleaned.join(", ")}
  end
end
```

Η παράμετρος cleaned είναι όλο το Array. Αν γράφαμε tap, το τελικό Hash θα αγνοούνταν και θα επέστρεφε το Array.

Η παραλλαγή

Χρησιμοποίησε yield_self για να βάλεις αγκύλες γύρω από ήδη ενωμένο κείμενο.

```ruby
["A", "B"].join(", ").yield_self { |text| "[#{text}]" }
```

Αποτέλεσμα: `"[A, B]"`.

066 / ΛΥΣΗ

Ποια μέθοδος ταιριάζει; · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{first: 4, total: 7}`.

```ruby
values = [1, 4, 2]
{first: values.find { |n| n > 2 }, total: values.sum}
```

Η κύρια άσκηση

```ruby
def stock_requests(stocks)
  {labels: stocks.map { |n| "#{n} τεμ." }, positive: stocks.select { |n| n > 0 },
first_large: stocks.find { |n| n >= 5 }, total: stocks.sum}
end
```

Το find επιστρέφει μία ποσότητα και μπορεί να δώσει nil. Η sum σε κενή λίστα δίνει 0. Διατηρούμε αυτές τις διαφορετικές συμβάσεις.

Η παραλλαγή

Βρες πόσες ποσότητες είναι τουλάχιστον 5 χωρίς ενδιάμεσο Array.

```ruby
[0, 7, 5, 2].count { |n| n >= 5 }
```

Αποτέλεσμα: `2`.

067 / ΛΥΣΗ

Αναφορά αποθήκης · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"x" => 5}`.

```ruby
[{category: "x", quantity: 2}, {category: "x", quantity: 3}].group_by { |p|
p[:category] }.transform_values { |rows| rows.sum { |p| p[:quantity] } }
```

Η κύρια άσκηση

```ruby
def warehouse_report(products)
  groups = products.select { |p| p[:quantity] > 0 }.group_by { |p| p[:category] }
  rows = groups.map do |category, entries|
    [category, entries.sum { |p| p[:quantity] }]
  end
  rows.sort_by { |row| row[0] }
end
```

Το φίλτρο προηγείται της group_by ώστε να μη δημιουργηθεί κατηγορία μηδενικού αποθέματος. Ταξινομούμε το μικρό τελικό αποτέλεσμα.

Η παραλλαγή

Μορφοποίησε ήδη έτοιμες γραμμές [["a", 3], ["b", 6]] σε κείμενο με newline.

```ruby
[["a", 3], ["b", 6]].map { |row| "#{row[0]}: #{row[1]}" }.join("\n")
```

Αποτέλεσμα: `"a: 3\nb: 6"`.

068 / ΛΥΣΗ

Κατάσταση μαζί με συμπεριφορά

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Seat 4"`.

```ruby
class SeatLabel
  def initialize(number)
    @number = number
  end
  def text
    "Seat #{@number}"
  end
end
SeatLabel.new(4).text
```

Η κύρια άσκηση

```ruby
class Booking
  def initialize(name, seats)
    @name = name
    @seats = seats
  end

  def description
    "#{@name}: #{@seats} θέσεις"
  end
end
```

Τα @name και @seats επιβιώνουν μετά το τέλος της initialize. Απλές τοπικές μεταβλητές name και seats δεν θα ήταν διαθέσιμες στην description.

Η παραλλαγή

Γράψε μικρή κλάση Ticket που κρατά price και quantity και δίνει total.

```ruby
class Ticket
  def initialize(price, quantity)
    @price = price
    @quantity = quantity
  end
  def total
    @price * @quantity
  end
end
Ticket.new(4, 3).total
```

Αποτέλεσμα: `12`.

069 / ΛΥΣΗ

Ανάγνωση και αλλαγή ιδιοτήτων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Ready"`.

```ruby
class Note
  attr_accessor :text
end
note = Note.new
note.text = "Ready"
note.text
```

Η κύρια άσκηση

```ruby
class Reservation
  attr_reader :id
  attr_accessor :status

  def initialize(id, status)
    @id = id
    @status = status
  end
end
```

Το id προστατεύεται επειδή δεν εκθέτουμε id=. Αυτό περιορίζει το δημόσιο API, δεν κάνει ολόκληρο το αντικείμενο immutable.

Η παραλλαγή

Δώσε μόνο setter για note και εμφάνισέ το μέσω ξεχωριστής description.

```ruby
class WrittenNote
  attr_writer :note
  def description
    "Note: #{@note}"
  end
end
r = WrittenNote.new
r.note = "Hi"
r.description
```

Αποτέλεσμα: `"Note: Hi"`.

070 / ΛΥΣΗ

Ο τρέχων receiver

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `0`.

```ruby
class Counter
  attr_accessor :value
  def reset
    self.value = 0
  end
end
c = Counter.new
c.reset
c.value
```

Η κύρια άσκηση

```ruby
class Stock
  attr_reader :quantity
  def initialize(quantity)
    self.quantity = quantity
  end
  def quantity=(value)
    @quantity = if value < 0
      0
    else
      value
    end
  end
  def add(amount)
    self.quantity = quantity + amount
    quantity
  end
end
```

Η add διαβάζει ξανά quantity στο τέλος, επειδή ο setter μπορεί να έχει περιορίσει την τιμή. Η ανάθεση από μόνη της θα επέστρεφε το αρχικό άθροισμα.

Η παραλλαγή

Δείξε ότι μια τοπική ανάθεση με ίδιο όνομα δεν καλεί setter.

RUBY

```ruby
class LocalCounter
  attr_accessor :value
  def try_local
    value = 9
    self.value
  end
end
c = LocalCounter.new
c.value = 2
c.try_local
```

Αποτέλεσμα: `2`.

071 / ΛΥΣΗ

Ονόματα με συμβάσεις

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `true`.

```ruby
class Switch
  def on?
    true
  end
end
Switch.new.on?
```

Η κύρια άσκηση

```ruby
class Toggle
  def initialize
    @active = false
  end
  def active?
    @active
  end
  def activate!
    @active = true
    self
  end
end
```

Επιστρέφουμε self επειδή το ορίζει η δική μας σύμβαση. Η κατάληξη ! δεν το εγγυάται, όπως είδαμε στη strip!.

Η παραλλαγή

Φτιάξε predicate empty? για δικό σου μικρό αντικείμενο που κρατά Array.

```ruby
class Bag
  def initialize(items)
    @items = items
  end
  def empty?
    @items.empty?
  end
end
Bag.new([]).empty?
```

Αποτέλεσμα: `true`.

072 / ΛΥΣΗ

Δημόσιο API και βοηθητικές μέθοδοι

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[OK]"`.

```ruby
class WrappedText
  def text
    "[#{content}]"
  end
  private
  def content
    "OK"
  end
end
WrappedText.new.text
```

Η κύρια άσκηση

```ruby
class Label
  def initialize(text)
    @text = text
  end
  def render
    "[#{normalized}]"
  end
  private
  def normalized
    @text.strip.upcase
  end
end
```

Η render καλεί normalized εσωτερικά. Η προστασία εκφράζει τι θέλουμε να εκθέσουμε, όχι μηχανισμό αποθήκευσης μυστικών.

Η παραλλαγή

Δύο αντικείμενα SecretRank συγκρίνουν εσωτερική protected rank, μέσω δημόσιας same_rank?.

```ruby
class SecretRank
  def initialize(rank)
    @rank = rank
  end
  def same_rank?(other)
    rank == other.rank
  end
  protected
  def rank
    @rank
  end
end
SecretRank.new(2).same_rank?(SecretRank.new(2))
```

Αποτέλεσμα: `true`.

073 / ΛΥΣΗ

Μέθοδοι της κλάσης

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `4`.

```ruby
class Defaults
  @size = 4
  def self.size
    @size
  end
end
Defaults.size
```

Η κύρια άσκηση

```ruby
class Room
  @default_capacity = 4
  def self.default_capacity
    @default_capacity
  end
  attr_reader :capacity
  def initialize(capacity = Room.default_capacity)
    @capacity = capacity
  end
end
```

Δεν χρησιμοποιούμε @@ μεταβλητή. Η class instance variable ανήκει στην Room και η @capacity σε κάθε room.

Η παραλλαγή

Δώσε class method label που επιστρέφει σταθερό κείμενο για όλη την κλάση.

```ruby
class Venue
  def self.label
    "Main hall"
  end
end
Venue.label
```

Αποτέλεσμα: `"Main hall"`.

074 / ΛΥΣΗ

Ονόματα σε δικό τους χώρο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"tea"`.

```ruby
module Kitchen
  DEFAULT = "tea"
end
Kitchen::DEFAULT
```

Η κύρια άσκηση

```ruby
module Catalog
  class Entry
    def initialize(name)
      @name = name
    end
    def label
      "Product: #{@name}"
    end
  end
end

module Guests
  class Entry
    def initialize(name)
      @name = name
    end
    def label
      "Guest: #{@name}"
    end
  end
end
```

Το namespace είναι μέρος της ταυτότητας του ονόματος. Catalog::Entry και Guests::Entry είναι διαφορετικές κλάσεις.

Η παραλλαγή

Όρισε DEFAULT_STATUS μέσα σε module Bookings.

```ruby
module Bookings
  DEFAULT_STATUS = "pending"
end
Bookings::DEFAULT_STATUS
```

Αποτέλεσμα: `"pending"`.

075 / ΛΥΣΗ

Δύο μικρά αντικείμενα · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[3, 1]`.

```ruby
class TinyBox
  attr_accessor :count
  def initialize
    @count = 1
  end
end
a = TinyBox.new
b = TinyBox.new
a.count = 3
[a.count, b.count]
```

Η κύρια άσκηση

```ruby
class SeatBooking
  def self.default_seats
    1
  end
  attr_reader :name, :seats
  def initialize(name, seats = SeatBooking.default_seats)
    @name = name
    self.seats = seats
  end
  def seats=(value)
    @seats = if value < 0
      0
    else
      value
    end
  end
end
```

Η initialize περνά από τον setter ώστε να ισχύει ο ίδιος κανόνας και κατά τη δημιουργία και κατά την αλλαγή.

Η παραλλαγή

Συγκέντρωσε ονόματα από δύο αντικείμενα με reader name.

```ruby
class NamedGuest
  attr_reader :name
  def initialize(name)
    @name = name
  end
end
[NamedGuest.new("Eva"), NamedGuest.new("Jo")].map { |g| g.name }
```

Αποτέλεσμα: `["Eva", "Jo"]`.

076 / ΛΥΣΗ

Αντικείμενα που συνεργάζονται

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"EVA"`.

```ruby
class UpperFormatter
  def format(name)
    name.upcase
  end
end
UpperFormatter.new.format("Eva")
```

Η κύρια άσκηση

```ruby
class PlainFormatter
  def format(name)
    name
  end
end
class BracketFormatter
  def format(name)
    "[#{name}]"
  end
end
class Report
  def initialize(formatter)
    @formatter = formatter
  end
  def lines(names)
    names.map { |name| @formatter.format(name) }
  end
end
```

Η Report δεν χρειάζεται if για κάθε είδος formatter. Καλεί μία κοινή μέθοδο και ο συνεργάτης ορίζει την παρουσίαση.

Η παραλλαγή

Φτιάξε τρίτο formatter που δίνει "Hi NAME".

```ruby
class GreetingFormatter
  def format(name)
    "Hi #{name}"
  end
end
GreetingFormatter.new.format("Jo")
```

Αποτέλεσμα: `"Hi Jo"`.

077 / ΛΥΣΗ

Εξειδίκευση υπάρχουσας συμπεριφοράς

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["basic", "special"]`.

```ruby
class BasicTag
  def label
    "basic"
  end
end
class SpecialTag < BasicTag
  def label
    "special"
  end
end
[BasicTag.new.label, SpecialTag.new.label]
```

Η κύρια άσκηση

```ruby
class StandardBooking
  def initialize(name)
    @name = name
  end
  def label
    "Booking #{@name}"
  end
end
class VipBooking < StandardBooking
  def label
    "VIP #{@name}"
  end
end
```

Η VipBooking δεν επαναλαμβάνει την initialize. Το override αλλάζει μόνο τη συμπεριφορά label που ζητήθηκε.

Η παραλλαγή

Δείξε κληρονόμηση μεθόδου χωρίς override.

RUBY

```ruby
class BaseCode
  def code
    "A1"
  end
end
class ChildCode < BaseCode
end
ChildCode.new.code
```

Αποτέλεσμα: `"A1"`.

078 / ΛΥΣΗ

Συνέχεια στη γονική υλοποίηση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hello!"`.

RUBY

```ruby
class BaseGreeting
  def text
    "Hello"
  end
end
class WarmGreeting < BaseGreeting
  def text
    "#{super()}!"
  end
end
WarmGreeting.new.text
```

Η κύρια άσκηση

RUBY

```ruby
class BasicReservation
  def initialize(name)
    @name = name
  end
  def label
    @name
  end
end
class NumberedReservation < BasicReservation
  def initialize(name, number)
    super(name)
    @number = number
  end
  def label
    "#{@number}: #{super()}"
  end
end
```

Στην initialize το σκέτο super θα προωθούσε και number και θα έδινε λάθος πλήθος ορισμάτων. Εκεί γράφουμε ρητά super(name).

Η παραλλαγή

Κάνε override μέθοδο με ίδιο όρισμα και προώθησέ το αυτόματα με σκέτο super.

RUBY

```ruby
class BaseEcho
  def echo(text)
    text
  end
end
class LoudEcho < BaseEcho
  def echo(text)
    super.upcase
  end
end
LoudEcho.new.echo("hi")
```

Αποτέλεσμα: `"HI"`.

079 / ΛΥΣΗ

Κοινή συμπεριφορά με module

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hi!"`.

```ruby
module Exclamation
  def shout(text)
    "#{text}!"
  end
end
class Speaker
  include Exclamation
end
Speaker.new.shout("Hi")
```

Η κύρια άσκηση

```ruby
module NamedLabel
  def label
    "[#{name}]"
  end
end
class PersonLabel
  include NamedLabel
  attr_reader :name
  def initialize(name)
    @name = name
  end
end
class ProductLabel
  include NamedLabel
  attr_reader :name
  def initialize(name)
    @name = name
  end
end
```

Η NamedLabel ζητά μόνο name. Το include προσθέτει instance methods, όχι μεθόδους που καλούνται πάνω στην ίδια την κλάση.

Η παραλλαγή

Μοιράσου predicate positive? που βασίζεται σε reader quantity.

RUBY

```ruby
module PositiveQuantity
  def positive?
    quantity > 0
  end
end
class UnitBox
  include PositiveQuantity
  def quantity
    2
  end
end
UnitBox.new.positive?
```

Αποτέλεσμα: `true`.

080 / ΛΥΣΗ

Μικρός κατάλογος κρατήσεων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva"]`.

RUBY

```ruby
class ActiveGuest
  attr_reader :name
  def initialize(name)
    @name = name
  end
  def active?
    true
  end
end
[ActiveGuest.new("Eva")].select { |g| g.active? }.map { |g| g.name }
```

Η κύρια άσκηση

```ruby
module BookingActivity
  def active?
    @active
  end
end
class CatalogBooking
  include BookingActivity
  attr_reader :name
  def initialize(name, active)
    @name = name
    @active = active
  end
end
class SimpleBookingFormat
  def format(name)
    name
  end
end
class FramedBookingFormat
  def format(name)
    "[#{name}]"
  end
end
class BookingCatalog
  def initialize(bookings)
    @bookings = bookings
  end
  def render(formatter)
    @bookings.select { |b| b.active? }.map { |b| formatter.format(b.name)
}.join("\n")
  end
end
```

Η BookingCatalog χρειάζεται μόνο τη μέθοδο format του συνεργάτη. Οι επιλογές εμφάνισης δεν αλλάζουν τις κρατήσεις.

Η παραλλαγή

Φτιάξε formatter που προσθέτει "Guest: " και δοκίμασέ τον μόνο του.

```ruby
class GuestPrefixFormat
  def format(name)
    "Guest: #{name}"
  end
end
GuestPrefixFormat.new.format("Jo")
```

Αποτέλεσμα: `"Guest: Jo"`.

081 / ΛΥΣΗ

Απόρριψη άκυρης κλήσης

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `6`.

```ruby
def positive_fee(n)
  raise ArgumentError, "positive required" if n <= 0
  n * 2
end
positive_fee(3)
```

Η κύρια άσκηση

```ruby
def checked_cost(seats, price)
  raise ArgumentError, "seats must be positive" if seats <= 0
  raise ArgumentError, "price must be nonnegative" if price < 0
  seats * price
end
```

Το raise δεν είναι ετικέτα αποτελέσματος. Ο καλών πρέπει να χειριστεί την εξαίρεση ή να την αφήσει να διαδοθεί.

Η παραλλαγή

Φτιάξε έλεγχο μήκους ονόματος: άδειο String προκαλεί ArgumentError, άλλο String επιστρέφεται.

```ruby
def required_name(name)
  raise ArgumentError, "name required" if name.empty?
  name
end
required_name("Eva")
```

Αποτέλεσμα: `"Eva"`.

082 / ΛΥΣΗ

Χειρισμός συγκεκριμένου σφάλματος

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `nil`.

```ruby
begin
  Integer("x", 10)
rescue ArgumentError
  nil
end
```

Η κύρια άσκηση

```ruby
def parse_seats(text)
  Integer(text, 10)
  rescue ArgumentError
    nil
end
```

Η rescue εδώ ανήκει στο σώμα της μεθόδου. Δεν πιάνουμε κάθε Exception: ένα προγραμματιστικό λάθος πρέπει να παραμείνει ορατό.

Η παραλλαγή

Χρησιμοποίησε rescue για να δώσεις το String "invalid" όταν αποτύχει αυστηρή μετατροπή.

```ruby
begin
  Integer("oops", 10)
rescue ArgumentError
  "invalid"
end
```

Αποτέλεσμα: `"invalid"`.

083 / ΛΥΣΗ

Το σφάλμα ως αντικείμενο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"missing title"`.

```ruby
begin
  raise ArgumentError, "missing title"
rescue ArgumentError => error
  error.message
end
```

Η κύρια άσκηση

```ruby
def validation_message(valid)
  raise ArgumentError, "invalid booking" unless valid
  {ok: true, message: "ready"}
  rescue ArgumentError => error
    {ok: false, message: error.message}
end
```

Η error υπάρχει μέσα στη rescue. Για πραγματικό απλό boolean κανόνα συνήθως αρκεί if· εδώ μαθαίνουμε την πρόσβαση στα στοιχεία μιας εξαίρεσης.

Η παραλλαγή

Μετά από raise, έλεγξε ότι το πρώτο στοιχείο backtrace είναι String.

```ruby
begin
  raise ArgumentError, "demo"
rescue ArgumentError => error
  error.backtrace.first.class == String
end
```

Αποτέλεσμα: `true`.

084 / ΛΥΣΗ

Τιμή από επιτυχία ή αποτυχία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"value=4"`.

```ruby
begin
  value = Integer("4", 10)
rescue ArgumentError
  "bad"
else
  "value=#{value}"
end
```

Η κύρια άσκηση

```ruby
def conversion_label(text)
  begin
    seats = Integer(text, 10)
  rescue ArgumentError
    "invalid"
  else
    "seats=#{seats}"
  end
end
```

Η μορφοποίηση ανήκει στο else και η αυστηρή μετατροπή στο begin. Έτσι η rescue δεν σκεπάζει άσχετες πράξεις.

Η παραλλαγή

Σε επιτυχημένη μετατροπή "0" επίστρεψε Hash {value: 0}, σε αποτυχία nil.

```ruby
begin
  value = Integer("0", 10)
rescue ArgumentError
  nil
else
  {value: value}
end
```

Αποτέλεσμα: `{value: 0}`.

085 / ΛΥΣΗ

Μικρός πίνακας αποτελεσμάτων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, nil]`.

```ruby
["2", "x"].map do |text|
  begin
    Integer(text, 10)
  rescue ArgumentError
    nil
  end
end
```

Η κύρια άσκηση

```ruby
def conversion_results(texts)
  texts.map do |text|
    begin
      value = Integer(text, 10)
    rescue ArgumentError
      {ok: false, value: nil}
    else
      {ok: true, value: value}
    end
  end
end
```

Η rescue βρίσκεται μέσα στη map ώστε μια άκυρη τιμή να μην ακυρώνει την υπόλοιπη λίστα.

Η παραλλαγή

Από έτοιμα conversion results μέτρησε τα ok.

```ruby
[{ok: true}, {ok: false}, {ok: true}].count { |r| r[:ok] }
```

Αποτέλεσμα: `2`.

086 / ΛΥΣΗ

Ενέργεια που γίνεται πάντα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[7, ["work", "cleanup"]]`.

```ruby
events = []
result = begin
  events << "work"
  7
ensure
  events << "cleanup"
end
[result, events]
```

Η κύρια άσκηση

```ruby
def tracked_operation(events, fail_now)
  events << "start"
  raise ArgumentError, "failed" if fail_now
  "done"
  ensure
    events << "cleanup"
end
```

Η ensure κάνει cleanup και αφήνει την αρχική επιστροφή ή εξαίρεση να συνεχίσει. Η τελευταία προσθήκη στο Array δεν γίνεται η επιστροφή της μεθόδου.

Η παραλλαγή

Παρατήρησε ensure μαζί με return από μέθοδο.

```ruby
def cleanup_return(events)
  begin
    return 4
  ensure
    events << "closed"
  end
end
events = []
[cleanup_return(events), events]
```

Αποτέλεσμα: `[4, ["closed"]]`.

087 / ΛΥΣΗ

Δικό σου είδος αποτυχίας

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"title required"`.

```ruby
class MissingTitle < StandardError
end
begin
  raise MissingTitle, "title required"
rescue MissingTitle => error
  error.message
end
```

Η κύρια άσκηση

```ruby
class InvalidQuantity < StandardError
end

def validated_quantity(quantity)
  raise InvalidQuantity, "negative quantity" if quantity < 0
  quantity
end
```

Κληρονομούμε από StandardError, όχι απευθείας Exception, για τη συνηθισμένη συμπεριφορά χειρισμού εφαρμογών.

Η παραλλαγή

Πιάσε συγκεκριμένη custom εξαίρεση και επίστρεψε false.

```ruby
class MissingCode < StandardError
end
begin
  raise MissingCode, "code required"
rescue MissingCode
  false
end
```

Αποτέλεσμα: `false`.

088 / ΛΥΣΗ

Απουσία αποτελέσματος χωρίς εξαίρεση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, 0]`.

```ruby
["2", "x", "0"].filter_map { |s| Integer(s, 10, exception: false) }
```

Η κύρια άσκηση

```ruby
def valid_integers(texts)
  texts.filter_map { |text| Integer(text, 10, exception: false) }
end
```

Η filter_map κρατά ο επειδή είναι truthy στη Ruby. Δεν χρειάζεται χειροκίνητη ειδική περίπτωση μηδενός.

Η παραλλαγή

Κράτησε μόνο μη αρνητικά από τις επιτυχημένες μετατροπές.

```ruby
["-2", "0", "x", "4"].filter_map { |s| Integer(s, 10, exception: false) }.select { |n| n >= 0 }
```

Αποτέλεσμα: `[0, 4]`.

089 / ΛΥΣΗ

Μικρός εισαγωγέας εγγραφών · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{name: "Eva", seats: 2}`.

```ruby
fields = "Eva:2".split(":", -1)
{name: fields[0], seats: Integer(fields[1], 10, exception: false)}
```

Η κύρια άσκηση

```ruby
def import_bookings(lines)
  lines.filter_map do |line|
    fields = line.split(":", -1)
    name = fields[0].strip
    seats = Integer(fields[1], 10, exception: false)
    if !name.empty? && !seats.nil? && seats > 0
      {name: name, seats: seats}
    end
  end
end
```

Το seats.nil? ελέγχεται πριν από seats > 0. Το split με -1 κρατά το τελικό κενό πεδίο, ώστε να απορριφθεί μέσω της μετατροπής.

Η παραλλαγή

Από ήδη έγκυρες εγγραφές υπολόγισε συνολικές θέσεις.

```ruby
[{name: "A", seats: 2}, {name: "B", seats: 3}].sum { |r| r[:seats] }
```

Αποτέλεσμα: `5`.

090 / ΛΥΣΗ

Κώδικας σε περισσότερα αρχεία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[true, false, "loaded"]`.

```ruby
first = require_relative "support/demo"
second = require_relative "support/demo"
[first, second, LoadDemo.text]
```

Η κύρια άσκηση

```ruby
def helper_greeting(name)
  require_relative "support/greeting"
  CourseGreeting.text(name)
end
```

Η require_relative δεν εξαρτάται από τον φάκελο όπου βρισκόταν το Terminal. Σε κανονικό αρχείο οι requires συνήθως συγκεντρώνονται στην αρχή· εδώ βρίσκονται στη μέθοδο για την άσκηση.

Η παραλλαγή

Με load φόρτωσε το ίδιο demo αρχείο μέσω πλήρους τοπικού path και κάλεσε τη μέθοδό του.

```ruby
load(__dir__ + "/support/demo.rb")
LoadDemo.text
```

Αποτέλεσμα: `"loaded"`.

091 / ΛΥΣΗ

Βιβλιοθήκες και εξαρτήσεις

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Array"`.

```ruby
Array.name
```

Η κύρια άσκηση

```ruby
def library_category(name)
  case name
  when "Array"
    "core"
  when "json"
    "default"
  when "csv"
    "bundled"
  when "rspec"
    "external"
  else
    nil
  end
end
```

Η κατηγοριοποίηση αναφέρεται στη συγκεκριμένη έκδοση Ruby. Το ότι ένα gem υπάρχει στο μηχάνημα δεν σημαίνει ότι ανήκει σε κάθε Bundler περιβάλλον.

Η παραλλαγή

Φόρτωσε JSON και επιβεβαίωσε ότι το όνομα του module είναι "JSON".

```ruby
require "json"
JSON.name
```

Αποτέλεσμα: `"JSON"`.

092 / ΛΥΣΗ

Γράψε δικό σου παράδειγμα συμπεριφοράς

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `true`.

```ruby
require "rspec/expectations"
include RSpec::Matchers
expect(2 + 3).to eq(5)
true
```

Η κύρια άσκηση

```ruby
require_relative "support/cost"

RSpec.describe "ticket_total" do
  it "υπολογίζει κανονική κράτηση" do
    expect(ticket_total(3, 4)).to eq(12)
  end
  it "δέχεται μηδενικές θέσεις" do
    expect(ticket_total(0, 4)).to eq(0)
  end
  it "δέχεται δωρεάν είσοδο" do
    expect(ticket_total(2, 0)).to eq(0)
  end
end
```

Οι αναμενόμενες τιμές είναι σταθερές από τη σύμβαση. Το expect(ticket_total(...)).to eq(ticket_total(...)) θα ήταν άχρηστο, γιατί συγκρίνει τη μέθοδο με τον εαυτό της.

Η παραλλαγή

Γράψε έναν έλεγχο ότι ένα κενό άθροισμα επιστρέφει 0.

```ruby
require "rspec/expectations"
include RSpec::Matchers
expect([].sum).to eq(0)
true
```

Αποτέλεσμα: `true`.

093 / ΛΥΣΗ

Έλεγχος αποτυχίας και μεταβολής

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `true`.

```ruby
require "rspec/expectations"
include RSpec::Matchers
a = [1]
b = a.dup
expect(b).to eq(a)
expect(b).not_to be(a)
true
```

Η κύρια άσκηση

```ruby
require_relative "support/names"

RSpec.describe "upper_names" do
  it "μετατρέπει χωρίς να αλλάζει την είσοδο" do
    names = ["Eva", "Jo"]
    result = upper_names(names)
    expect(result).to eq(["EVA", "JO"])
    expect(names).to eq(["Eva", "Jo"])
    expect(result).not_to be(names)
  end
  it "δίνει νέο κενό Array" do
    names = []
    result = upper_names(names)
    expect(result).to eq([])
    expect(result).not_to be(names)
  end
  it "απορρίπτει την απουσία λίστας" do
    expect { upper_names(nil) }.to raise_error(ArgumentError, "names required")
  end
end
```

Η eq δεν αποδεικνύει νέα αναφορά. Επίσης η σύγκριση names με ένα δεύτερο όνομα του ίδιου Array δεν αποδεικνύει ότι δεν άλλαξε.

Η παραλλαγή

Έλεγξε ότι η raise μεταδίδει και τη σωστή κλάση και το μήνυμα.

```ruby
require "rspec/expectations"
include RSpec::Matchers
expect { raise ArgumentError, "bad" }.to raise_error(ArgumentError, "bad")
true
```

Αποτέλεσμα: `true`.

094 / ΛΥΣΗ

Ανάγνωση ολόκληρου κειμένου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `3`.

```ruby
File.read(__dir__ + "/fixtures/names.txt", encoding: "UTF-8").lines.length
```

Η κύρια άσκηση

```ruby
def read_names(path)
  File.read(path, encoding: "UTF-8").lines.map { |line| line.strip }.reject { |line| line.empty? }
end
```

Η read ταιριάζει σε μικρό δοσμένο αρχείο. Η επεξεργασία Strings γίνεται μετά την ανάγνωση και δεν μεταβάλλει το αρχείο.

Η παραλλαγή

Διάβασε το ίδιο fixture και κράτησε τις γραμμές με chomp, χωρίς αφαίρεση κενών.

```ruby
File.read(__dir__ + "/fixtures/names.txt", encoding: "UTF-8").lines.map { |line| line.chomp }
```

Αποτέλεσμα: `[" Εύα ", "", "Άρης"]`.

095 / ΛΥΣΗ

Άνοιγμα με ορισμένο χρόνο ζωής

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Ready"`.

```ruby
File.open(__dir__ + "/fixtures/message.txt", "r:UTF-8") { |file| file.read.chomp }
```

Η κύρια άσκηση

```ruby
def read_and_track(path, audit)
  File.open(path, "r:UTF-8") do |file|
    audit << file
    file.read
  end
end
```

Το block μορφοποιεί τη διάρκεια ζωής του File. Αν καλούσαμε open χωρίς block, θα έπρεπε να οργανώσουμε ρητά το close.

Η παραλλαγή

Κάνε raise μέσα σε File.open block και επιβεβαίωσε ότι το File έκλεισε.

```ruby
file = nil
begin
  File.open(__dir__ + "/fixtures/message.txt", "r:UTF-8") do |opened|
    file = opened
    raise ArgumentError, "stop"
  end
rescue ArgumentError
  file.closed?
end
```

Αποτέλεσμα: `true`.

096 / ΛΥΣΗ

Γραμμές μία μία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `3`.

```ruby
count = 0
File.foreach(__dir__ + "/fixtures/log.txt", encoding: "UTF-8") { |line| count =
count + 1 }
count
```

Η κύρια άσκηση

```ruby
def error_lines(path)
  result = []
  File.foreach(path, encoding: "UTF-8") do |line|
    result << line.chomp if line.start_with?("ERROR:")
  end
  result
end
```

Επιστρέφουμε result μετά τη foreach. Η foreach με block δεν συγκεντρώνει από μόνη της τις επιστροφές του block.

Η παραλλαγή

Με foreach μέτρησε γραμμές που αρχίζουν INFO:.

```ruby
count = 0
File.foreach(__dir__ + "/fixtures/log.txt", encoding: "UTF-8") do |line|
  count = count + 1 if line.start_with?("INFO:")
end
count
```

Αποτέλεσμα: `1`.

097 / ΛΥΣΗ

Παραγωγή αρχείου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"A\nB\n"`.

```ruby
path = __dir__ + "/tmp/example.txt"
File.write(path, "A\n", mode: "w:UTF-8")
File.write(path, "B\n", mode: "a:UTF-8")
File.read(path, encoding: "UTF-8")
```

Η κύρια άσκηση

```ruby
def save_lines(path, lines, append: false)
  text = lines.map { |line| "#{line}\n" }.join
  mode = if append
    "a:UTF-8"
  else
    "w:UTF-8"
  end
  File.write(path, text, mode: mode)
  path
end
```

Το w αδειάζει το υπάρχον περιεχόμενο. Τα tests δίνουν ξεχωριστούς προσωρινούς φακέλους μέσα στο μάθημα και επιβεβαιώνουν χωριστά append και αντικατάσταση.

Η παραλλαγή

Γράψε μία ελληνική γραμμή σε tmp/variation.txt και διάβασέ την πίσω.

```ruby
path = __dir__ + "/tmp/variation.txt"
File.write(path, "Γεια\n", mode: "w:UTF-8")
File.read(path, encoding: "UTF-8")
```

Αποτέλεσμα: `"Γεια\n"`.

098 / ΛΥΣΗ

Διαδρομές και επιλογή αρχείων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["day.csv", ".csv"]`.

```ruby
require "pathname"
path = Pathname.new("reports/day.csv")
[path.basename.to_s, path.extname]
```

Η κύρια άσκηση

```ruby
def text_filenames(directory)
  Dir.glob(File.join(directory, "*.txt")).map { |path| File.basename(path) }.sort
end
```

Η glob επιστρέφει paths. Η basename κρατά μόνο το όνομα που ζητά η αναφορά. Το μοτίβο δεν κάνει αναδρομή σε υποφακέλους.

Η παραλλαγή

Σύνθεσε path από τρία μέρη χωρίς χειροκίνητα slash.

```ruby
File.join("reports", "2026", "day.csv")
```

Αποτέλεσμα: `"reports/2026/day.csv"`.

099 / ΛΥΣΗ

Ορίσματα από το Terminal

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{input: "names.txt", output: "report.txt"}`.

```ruby
args = ["names.txt", "report.txt"]
{input: args[0], output: args[1]}
```

Η κύρια άσκηση

```ruby
def command_paths(args = ARGV)
  raise ArgumentError, "expected input and output" unless args.length == 2
  {input: args[0], output: args[1]}
end
```

Η μέθοδος δέχεται και κανονικά Arrays για έλεγχο. Δεν χρησιμοποιεί shift, άρα δεν καταναλώνει τα ορίσματα του καλούντος.

Η παραλλαγή

Από δοσμένο Array args διάβασε προαιρετικό τρίτο όρισμα με default "plain".

```ruby
args = ["a", "b"]
args[2] || "plain"
```

Αποτέλεσμα: `"plain"`.

100 / ΛΥΣΗ

Ροές εισόδου και εξόδου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Eva"`.

```ruby
require "stringio"
input = StringIO.new("Eva\n")
input.gets.chomp
```

Η κύρια άσκηση

```ruby
def greet_stream(input = STDIN, output = STDOUT, errors = STDERR)
  line = input.gets
  if line.nil? || line.strip.empty?
    errors.puts("name required")
    false
  else
    output.puts("Hello #{line.strip}")
    true
  end
end
```

Χρησιμοποιούμε τις δοσμένες ροές αντί για σκέτη puts, ώστε το αποτέλεσμα να είναι ελέγξιμο και η έξοδος σφαλμάτων χωριστή.

Η παραλλαγή

Γράψε δύο γραμμές σε StringIO και δώσε το τελικό String.

```ruby
require "stringio"
out = StringIO.new
out.puts("A")
out.puts("B")
out.string
```

Αποτέλεσμα: `"A\nB\n"`.

101 / ΛΥΣΗ

JSON

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"name" => "Eva", "active" => false, "note" => nil}`.

```ruby
require "json"
JSON.parse('{"name":"Eva","active":false,"note":null}')
```

Η κύρια άσκηση

```ruby
def json_names(text)
  require "json"
  records = JSON.parse(text)
  names = records.select { |record| record["active"] }.map { |record| record["name"]
}
  JSON.generate(names)
end
```

Η πρόσβαση record[:active] δεν διαβάζει το "active" που έδωσε η JSON.parse. Κράτησε συνεπή τύπο κλειδιών.

Η παραλλαγή

Κάνε generate και parse ενός Hash με false και nil και δες τα String κλειδιά.

```ruby
require "json"
JSON.parse(JSON.generate({active: false, note: nil}))
```

Αποτέλεσμα: `{"active" => false, "note" => nil}`.

102 / ΛΥΣΗ

CSV

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Eva, Jr"`.

```ruby
require "csv"
CSV.parse("name,seats\n\"Eva, Jr\",2\n", headers: true).first["name"]
```

Η κύρια άσκηση

```ruby
def csv_bookings(text)
  require "csv"
  CSV.parse(text, headers: true).map do |row|
    {name: row["name"], seats: Integer(row["seats"], 10)}
  end
end
```

Τα CSV πεδία παραμένουν Strings μέχρι να τα μετατρέψεις. Τα headers επιτρέπουν πρόσβαση με όνομα, χωρίς εξάρτηση από αριθμό στήλης.

Η παραλλαγή

Διάβασε όνομα που περιέχει escaped διπλό εισαγωγικό από CSV.

```ruby
require "csv"
CSV.parse("name,seats\n\"Eva \"\"E\"\"\",1\n", headers: true).first["name"]
```

Αποτέλεσμα: `'Eva "E"'`.

103 / ΛΥΣΗ

Ρυθμίσεις από το περιβάλλον

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `0`.

```ruby
demo_env = {"RUBY_LESSON_LIMIT" => "0"}
Integer(demo_env.fetch("RUBY_LESSON_LIMIT", "10"), 10)
```

Η κύρια άσκηση

```ruby
def demo_settings(env = ENV)
  {label: env.fetch("RUBY_LESSON_LABEL", "Guest"), limit:
Integer(env.fetch("RUBY_LESSON_LIMIT", "10"), 10)}
end
```

Το String "0" είναι truthy, άρα δεν είναι boolean false. Οι μετατροπές ρυθμίσεων πρέπει να είναι ρητές.

Η παραλλαγή

Από δοσμένο Hash διάβασε επινοημένο flag: ενεργό μόνο όταν η τιμή είναι ακριβώς "1".

```ruby
demo_env = {"RUBY_LESSON_ACTIVE" => "0"}
demo_env["RUBY_LESSON_ACTIVE"] == "1"
```

Αποτέλεσμα: `false`.

104 / ΛΥΣΗ

Αναφορά από fixtures · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"Eva" => 2, "Jo" => 1}`.

```ruby
File.read(__dir__ + "/fixtures/names.txt", encoding: "UTF-8").lines.map { |s| s.strip }.reject { |s| s.empty? }.tally
```

Η κύρια άσκηση

```ruby
def write_name_counts(input_path, output_path)
  counts = File.read(input_path, encoding: "UTF-8").lines.map { |s| s.strip }.reject { |s| s.empty? }.tally
  text = counts.sort_by { |name, count| name }.map { |name, count| "#{name}: #{count}\n" }.join
  File.write(output_path, text, mode: "w:UTF-8")
  counts.length
end
```

Η μέθοδος επιστρέφει πόσα διαφορετικά ονόματα υπάρχουν, όχι bytes αρχείου και όχι το άθροισμα εμφανίσεων.

Η παραλλαγή

Από έτοιμα πλήθη δώσε το συνολικό πλήθος συμμετοχών.

```ruby
{"Eva" => 2, "Jo" => 1}.values.sum
```

Αποτέλεσμα: `3`.

105 / ΛΥΣΗ

Τοπικό εργαλείο αναφορών · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `'[{"name":"Eva","seats":2}]'`.

```ruby
require "csv"
require "json"
rows = CSV.parse("name,seats\nEva,2\n", headers: true)
JSON.generate(rows.map { |row| {name: row["name"], seats: Integer(row["seats"], 10)}
})
```

Η κύρια άσκηση

```ruby
def convert_booking_file(input_path, output_path)
  require "csv"
  require "json"
  text = File.read(input_path, encoding: "UTF-8")
  records = CSV.parse(text, headers: true).filter_map do |row|
    name = row["name"]&.strip
    seats = Integer(row["seats"], 10, exception: false)
    if name && !name.empty? && seats && seats > 0
      {name: name, seats: seats}
    end
  end
  File.write(output_path, JSON.generate(records), mode: "w:UTF-8")
  records.length
end
```

Η CSV χειρίζεται το κόμμα μέσα σε quoted όνομα και η JSON.generate την κωδικοποίηση της εξόδου. Ο δικός μας κώδικας περιορίζεται στους κανόνες πεδίων.

Η παραλλαγή

Από έτοιμο JSON Array εγγραφών πάρε συνολικές θέσεις.

```ruby
require "json"
JSON.parse('[{"name":"Eva","seats":2},{"name":"Jo","seats":3}]').sum { |r|
r["seats"] }
```

Αποτέλεσμα: `5`.

106 / ΛΥΣΗ

Η δική σου μέθοδος δέχεται block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[EVA]"`.

```ruby
def surround(value)
  "[#{yield(value)}]"
end
surround("Eva") { |name| name.upcase }
```

Η κύρια άσκηση

```ruby
def format_lines(lines)
  lines.map { |line| yield(line) }
end
```

Η map συγκεντρώνει τις επιστροφές της yield. Δεν χρειάζεται να γνωρίζει ποια μορφοποίηση διάλεξε ο καλών.

Η παραλλαγή

Δώσε δύο τιμές στο ίδιο block και πάρε το άθροισμά τους.

```ruby
def combine_pair(a, b)
  yield(a, b)
end
combine_pair(2, 3) { |a, b| a + b }
```

Αποτέλεσμα: `5`.

107 / ΛΥΣΗ

Το block είναι προαιρετικό

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hi Eva"`.

```ruby
def greet_optional(name)
  return yield(name) if block_given?
  "Hi #{name}"
end
greet_optional("Eva")
```

Η κύρια άσκηση

```ruby
def flexible_label(name)
  return yield(name) if block_given?
  "Guest #{name}"
end
```

Η παρουσία block ελέγχεται με block_given?, όχι από την αλήθεια του αποτελέσματός του.

Η παραλλαγή

Κάλεσε προαιρετικό block δύο φορές όταν υπάρχει, αλλιώς δώσε [].

```ruby
def twice_optional(value)
  return [] unless block_given?
  [yield(value), yield(value)]
end
twice_optional(3) { |n| n + 1 }
```

Αποτέλεσμα: `[4, 4]`.

108 / ΛΥΣΗ

Block ως τιμή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[Jo]"`.

```ruby
formatter = proc { |name| "[#{name}]" }
formatter.call("Jo")
```

Η κύρια άσκηση

```ruby
def capture_formatter(&block)
  block
end

def apply_formatter(name, formatters, index)
  formatters[index].call(name)
end
```

Το & χρησιμοποιείται στην υπογραφή για σύλληψη του block. Μέσα στο σώμα το block είναι κανονική τοπική μεταβλητή που δέχεται call.

Η παραλλαγή

Αποθήκευσε proc για διπλασιασμό και κάλεσέ το με 4.

```ruby
double = proc { |number| number * 2 }
double.call(4)
```

Αποτέλεσμα: `8`.

109 / ΛΥΣΗ

Μεταβλητές που θυμάται ένα block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hello Eva"`.

```ruby
prefix = "Hi"
f = proc { |name| "#{prefix} #{name}" }
prefix = "Hello"
f.call("Eva")
```

Η κύρια άσκηση

```ruby
def prefix_formatter(prefix)
  proc { |name| "#{prefix} #{name}" }
end
```

Δεν χρειάζεται global μεταβλητή. Το prefix ανήκει στο περιβάλλον της συγκεκριμένης κλήσης κατασκευής.

Η παραλλαγή

Φτιάξε closure που αυξάνει ιδιωτικό τοπικό μετρητή σε κάθε call.

```ruby
def counter_proc
  count = 0
  proc { count = count + 1 }
end
counter = counter_proc
[counter.call, counter.call]
```

Αποτέλεσμα: `[1, 2]`.

110 / ΛΥΣΗ

Συνάρτηση με δικά της όρια

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[6, 1]`.

```ruby
double = ->(n) { n * 2 }
[double.call(3), double.arity]
```

Η κύρια άσκηση

```ruby
def price_calculator(price)
  ->(quantity) { quantity * price }
end
```

Η λογική της χρέωσης είναι μία πράξη. Το νέο στοιχείο είναι η σύμβαση κλήσης της lambda.

Η παραλλαγή

Γράψε την ίδια ιδέα με τη λέξη lambda και πρόσθεση αντί για γινόμενο.

```ruby
add_fee = lambda { |amount| amount + 2 }
add_fee.call(5)
```

Αποτέλεσμα: `7`.

111 / ΛΥΣΗ

Από πού επιστρέφει το return;

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["inside", "after"]`.

```ruby
def lambda_boundary
  action = -> { return "inside" }
  [action.call, "after"]
end
lambda_boundary
```

Η κύρια άσκηση

```ruby
def lambda_flow
  action = -> { return "inside" }
  [action.call, "after"]
end

def proc_flow
  action = proc { return "inside" }
  action.call
  "after"
end
```

Οι δύο callables μοιάζουν ως αντικείμενα αλλά το return έχει διαφορετικό στόχο. Στην proc_flow η γραμμή "after" δεν εκτελείται.

Η παραλλαγή

Επίστρεψε proc με return από factory και πιάσε το LocalJumpError όταν το καλέσεις μετά.

```ruby
def escaped_proc
  proc { return "inside" }
end
begin
  escaped_proc.call
rescue LocalJumpError
  "expired return target"
end
```

Αποτέλεσμα: `"expired return target"`.

112 / ΛΥΣΗ

Callable με προσαρμογή · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hi!"`.

```ruby
def suffixer(suffix)
  ->(text) { "#{text}#{suffix}" }
end
suffixer("!").call("Hi")
```

Η κύρια άσκηση

```ruby
def custom_formatter(prefix, uppercase: false)
  ->(name) do
    text = if uppercase
      name.upcase
    else
      name
    end
    "#{prefix} #{text}"
  end
end
```

Η τελική έκφραση της lambda αρκεί για επιστροφή. Δεν χρειάζεται return που θα αύξανε το νοητικό φορτίο χωρίς όφελος.

Η παραλλαγή

Φτιάξε lambda που θυμάται άνοιγμα και κλείσιμο ετικέτας.

```ruby
def wrapper(left, right)
  ->(text) { "#{left}#{text}#{right}" }
end
wrapper("[", "]").call("Eva")
```

Αποτέλεσμα: `"[Eva]"`.

113 / ΛΥΣΗ

Υπάρχουσα μέθοδος ως callable

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"EVA"`.

```ruby
formatter = "Eva".method(:upcase)
formatter.call
```

Η κύρια άσκηση

```ruby
class TextFormats
  def upper(name)
    name.upcase
  end
  def bracket(name)
    "[#{name}]"
  end
  def selected(style)
    case style
    when :upper
      method(:upper)
    when :bracket
      method(:bracket)
    else
      raise ArgumentError, "unknown style"
    end
  end
end
```

Το Method θυμάται και ποια μέθοδο και σε ποιο αντικείμενο θα την καλέσει. Δεν είναι απλώς το Symbol του ονόματος.

Η παραλλαγή

Κράτησε Method του length δεμένο στο String "Ruby" και κάλεσέ το.

```ruby
size = "Ruby".method(:length)
size.call
```

Αποτέλεσμα: `4`.

114 / ΛΥΣΗ

Μετατροπή σε block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["EVA", "JO"]`.

```ruby
["Eva", "Jo"].map(&:upcase)
```

Η κύρια άσκηση

```ruby
def shorthand_names(names)
  names.map(&:strip).map(&:upcase)
end
```

Η map(&:strip.upcase) δεν συνθέτει κλήσεις πάνω στο στοιχείο. Για σύνθετη λογική γράψε κανονικό block ή ξεχωριστή callable.

Η παραλλαγή

Πέρασε αποθηκευμένη lambda ως block της map.

```ruby
format = ->(name) { "[#{name}]" }
["Eva", "Jo"].map(&format)
```

Αποτέλεσμα: `["[Eva]", "[Jo]"]`.

115 / ΛΥΣΗ

Μια αλυσίδα που εξηγείται · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"A"`.

```ruby
[" A ", " "].map(&:strip).reject(&:empty?).then { |names| names.join(", ") }
```

Η κύρια άσκηση

```ruby
def formatted_report(names, audit, formatter)
  names.map(&:strip).reject(&:empty?).tap { |cleaned| audit << cleaned.length }.then
{ |cleaned| formatter.call(cleaned) }
end
```

Η έξοδος του formatter δεν χρειάζεται να είναι String. Η then διατηρεί και false ως νόμιμη τελική τιμή.

Η παραλλαγή

Με tap κράτησε την έτοιμη λίστα σε audit και με then δώσε το μήκος της.

```ruby
audit = []
value = ["A", "B"].tap { |names| audit << names }.then(&:length)
[value, audit]
```

Αποτέλεσμα: `[2, [["A", "B"]]]`.

116 / ΛΥΣΗ

Μεταβλητό πλήθος ορισμάτων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Eva, Jo"`.

```ruby
def joined_names(*names)
  names.join(", ")
end
joined_names("Eva", "Jo")
```

Η κύρια άσκηση

```ruby
def message_lines(title, *lines)
  ([title] + lines).join("\n")
end
```

Το lines μέσα στο σώμα είναι Array. Ο αστερίσκος βρίσκεται στον ορισμό για συλλογή ορισμάτων.

Η παραλλαγή

Φτιάξε μέθοδο που δέχεται όσα ποσά θέλεις και τα αθροίζει.

```ruby
def sum_amounts(*amounts)
  amounts.sum
end
sum_amounts(2, 3, 4)
```

Αποτέλεσμα: `9`.

117 / ΛΥΣΗ

Άνοιγμα λίστας στην κλήση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[1, 2, 3, 4]`.

```ruby
values = [2, 3]
[1, *values, 4]
```

Η κύρια άσκηση

```ruby
def pair_label(left, right)
  "#{left}/#{right}"
end

def expanded_label(pair)
  pair_label(*pair)
end
```

Η pair_label(pair) θα έδινε ένα μόνο όρισμα, που είναι Array. Η pair_label(*pair) δίνει τα δύο στοιχεία χωριστά.

Η παραλλαγή

Πρόσθεσε αρχική και τελική ετικέτα γύρω από Array ονομάτων με splat literal.

```ruby
names = ["Eva", "Jo"]
["start", *names, "end"]
```

Αποτέλεσμα: `["start", "Eva", "Jo", "end"]`.

118 / ΛΥΣΗ

Ονομασμένες επιλογές ως hash

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"<A>"`.

```ruby
def box_text(text, left: "[", right: "]")
  "#{left}#{text}#{right}"
end
options = {left: "<", right: ">"}
box_text("A", **options)
```

Η κύρια άσκηση

```ruby
def heading(name, prefix: "", suffix: "!")
  "#{prefix}#{name}#{suffix}"
end

def forwarded_heading(name, **options)
  heading(name, **options)
end
```

Ένα μονό * ανοίγει positional ορίσματα. Εδώ χρειάζεται ** για τη συμφωνία keywords.

Η παραλλαγή

Συγκέντρωσε keywords σε Hash και επίστρεψέ τα.

```ruby
def collect_options(**options)
  options
end
collect_options(color: "blue", size: 2)
```

Αποτέλεσμα: `{color: "blue", size: 2}`.

119 / ΛΥΣΗ

Η ίδια μέθοδος με διαφορετικές κλήσεις · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Guest Eva: 3"`.

```ruby
def count_label(name, quantity = 1, prefix: "")
  "#{prefix}#{name}: #{quantity}"
end
count_label(*["Eva", 3], **{prefix: "Guest "})
```

Η κύρια άσκηση

```ruby
def count_label(name, quantity = 1, prefix: "")
  "#{prefix}#{name}: #{quantity}"
end

def call_from_data(arguments, options)
  count_label(*arguments, **options)
end
```

Δεν ενώνουμε arguments και options σε ένα Array. Το τελευταίο Hash είναι keywords μόνο όταν το ανοίγουμε ρητά με **.

Η παραλλαγή

Κάλεσε τη μικρή μέθοδο format_pair(a, b:) με ένα positional και ένα keyword.

```ruby
def format_pair(a, b:)
  "#{a}/#{b}"
end
format_pair(*["X"], **{b: "Y"})
```

Αποτέλεσμα: `"X/Y"`.

120 / ΛΥΣΗ

Ανάθεση σε πολλά ονόματα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", ["B", "C"], "D"]`.

```ruby
first, *middle, last = ["A", "B", "C", "D"]
[first, middle, last]
```

Η κύρια άσκηση

```ruby
def route_parts(stops)
  first, *middle, last = stops
  {first: first, middle: middle, last: last}
end
```

Το *middle είναι Array ακόμη και όταν δεν υπάρχει ενδιάμεση στάση. Η αρχική συλλογή δεν καταναλώνεται.

Η παραλλαγή

Αντάλλαξε δύο τοπικές τιμές με πολλαπλή ανάθεση.

```ruby
left = "A"
right = "B"
left, right = right, left
[left, right]
```

Αποτέλεσμα: `["B", "A"]`.

121 / ΛΥΣΗ

Δομή παραμέτρων block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva: 2", "Jo: 1"]`.

```ruby
[["Eva", 2], ["Jo", 1]].map { |name, seats| "#{name}: #{seats}" }
```

Η κύρια άσκηση

```ruby
def nested_labels(records)
  records.map { |(name, seats), paid| "#{name}: #{seats} / #{paid}" }
end
```

Οι παρενθέσεις δηλώνουν ποια εσωτερική δομή ανοίγεται. Δεν είναι κλήση μεθόδου.

Η παραλλαγή

Από ζευγάρια ονόματος και ποσότητας κράτησε μόνο τα ονόματα με destructuring.

```ruby
[["Eva", 2], ["Jo", 1]].map { |name, seats| name }
```

Αποτέλεσμα: `["Eva", "Jo"]`.

122 / ΛΥΣΗ

Formatter με επιλογές · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"* A\n* B"`.

```ruby
def decorate(text, prefix: "")
  "#{prefix}#{text}"
end
["A", "B"].map { |text| decorate(text, **{prefix: "* "}) }.join("\n")
```

Η κύρια άσκηση

```ruby
def format_row(name, seats, prefix: "", uppercase: false)
  name = name.upcase if uppercase
  "#{prefix}#{name}: #{seats}"
end

def format_table(rows, **options)
  rows.map { |name, seats| format_row(name, seats, **options) }.join("\n")
end
```

Η επιλογή uppercase επηρεάζει μόνο το όνομα. Η αναφορά παραμένει υπεύθυνη για την ένωση γραμμών.

Η παραλλαγή

Μορφοποίησε ζευγάρια με δική σου lambda μέσω map.

```ruby
formatter = ->(pair) { "#{pair[0]}=#{pair[1]}" }
[["A", 2], ["B", 3]].map(&formatter)
```

Αποτέλεσμα: `["A=2", "B=3"]`.

123 / ΛΥΣΗ

Προεπιλεγμένη τιμή hash

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[0, false]`.

```ruby
counts = Hash.new(0)
value = counts[:missing]
[value, counts.key?(:missing)]
```

Η κύρια άσκηση

```ruby
def manual_counts(categories)
  counts = Hash.new(0)
  categories.each { |category| counts[category] += 1 }
  counts
end
```

Η ανάγνωση default δεν αποθηκεύει τίποτα. Η ανάθεση μέσα στο += δημιουργεί το κλειδί.

Η παραλλαγή

Ξεκίνα από default 10 και πρόσθεσε 2 σε νέο κλειδί.

```ruby
totals = Hash.new(10)
totals[:a] += 2
[totals[:a], totals[:b], totals.keys]
```

Αποτέλεσμα: `[12, 10, [:a]]`.

124 / ΛΥΣΗ

Προεπιλογή που κατασκευάζεται

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `{"Eva" => ["call"], "Jo" => ["email"]}`.

```ruby
notes = Hash.new { |hash, key| hash[key] = [] }
notes["Eva"] << "call"
notes["Jo"] << "email"
notes
```

Η κύρια άσκηση

```ruby
def notes_by_name(entries)
  notes = Hash.new { |hash, key| hash[key] = [] }
  entries.each { |name, note| notes[name] << note }
  notes
end
```

Το κρίσιμο βήμα είναι η ανάθεση hash[key] = []. Η επιστροφή [] χωρίς ανάθεση θα κατασκεύαζε προσωρινή λίστα σε κάθε ανάγνωση.

Η παραλλαγή

Δείξε το πρόβλημα ενός κοινού default Array.

```ruby
h = Hash.new([])
h[:a] << "x"
[h[:b], h.keys]
```

Αποτέλεσμα: `[["x"], []]`.

125 / ΛΥΣΗ

Αποτέλεσμα που ενημερώνεται

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["[A]", "[B]"]`.

```ruby
["A", "B"].each_with_object([]) { |name, result| result << "[#{name}]" }
```

Η κύρια άσκηση

```ruby
def products_index(products)
  products.each_with_object({}) { |product, index| index[product[:code]] = product }
end
```

Η επαναανάθεση index = {} μέσα στο block δεν θα άλλαζε ποιο αντικείμενο επιστρέφεται. Μεταβάλλουμε το δοσμένο Hash.

Η παραλλαγή

Χτίσε Array με τα θετικά στοιχεία χρησιμοποιώντας each_with_object.

```ruby
[-1, 0, 2, 3].each_with_object([]) { |n, result| result << n if n > 0 }
```

Αποτέλεσμα: `[2, 3]`.

126 / ΛΥΣΗ

Αποτέλεσμα που περνά στο επόμενο βήμα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `5`.

```ruby
[2, 3].reduce(10) { |balance, amount| balance - amount }
```

Η κύρια άσκηση

```ruby
def apply_changes(initial, changes)
  changes.reduce(initial) do |balance, (operation, value)|
    case operation
    when "add"
      balance + value
    when "multiply"
      balance * value
    end
  end
end
```

Η reduce χρησιμοποιεί την επιστροφή του block στο επόμενο βήμα. Η each_with_object δεν κάνει το ίδιο, γι' αυτό ένα απλό += σε αριθμητικό accumulator δεν είναι ισοδύναμο.

Η παραλλαγή

Με inject ένωσε τρία Strings με αρχικό κενό κείμενο.

```ruby
["A", "B", "C"].inject("") { |text, part| text + part }
```

Αποτέλεσμα: `"ABC"`.

127 / ΛΥΣΗ

Ανεξάρτητες αρχικές τιμές

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["read"], []]`.

```ruby
days = Array.new(2) { [] }
days[0] << "read"
days
```

Η κύρια άσκηση

```ruby
def day_lists(count)
  Array.new(count) { [] }
end
```

Το block δημιουργεί αντικείμενα, δεν επαναλαμβάνει μία ήδη κατασκευασμένη αναφορά.

Η παραλλαγή

Χρησιμοποίησε τον index του block για ετικέτες Day 1 έως Day 3.

```ruby
Array.new(3) { |index| "Day #{index + 1}" }
```

Αποτέλεσμα: `["Day 1", "Day 2", "Day 3"]`.

128 / ΛΥΣΗ

Μνήμη προηγούμενου αποτελέσματος

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `false`.

```ruby
cache = {answer: false}
if cache.key?(:answer)
  cache[:answer]
else
  "compute"
end
```

Η κύρια άσκηση

```ruby
class MemoCheck
  def initialize(&check)
    @check = check
    @computed = false
  end
  def value
    return @value if @computed
    @value = @check.call
    @computed = true
    @value
  end
end
```

Το @computed γίνεται true μόνο αφού τελειώσει η callable. Έτσι μια εξαίρεση δεν σημειώνεται ως ολοκληρωμένος υπολογισμός.

Η παραλλαγή

Χρησιμοποίησε Hash key? για cache όπου το αποθηκευμένο αποτέλεσμα είναι nil.

```ruby
cache = {}
calls = 0
[1, 2].each do
  unless cache.key?(:result)
    calls += 1
    cache[:result] = nil
  end
end
[cache[:result], calls]
```

Αποτέλεσμα: `[nil, 1]`.

129 / ΛΥΣΗ

Έλεγχος της επανάληψης

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"found 4"`.

```ruby
[1, 4, 2].each do |n|
  break "found #{n}" if n >= 3
end
```

Η κύρια άσκηση

```ruby
def names_until_stop(lines)
  names = []
  lines.each do |line|
    name = line.strip
    next if name.empty?
    break if name == "STOP"
    names << name
  end
  names
end
```

Η μέθοδος επιστρέφει names μετά την each. Το break από μόνο του δεν επιστρέφει αυτόματα τον συσσωρευτή.

Η παραλλαγή

Με map δώσε "skip" για το 0 με next και διπλάσιο για άλλους αριθμούς.

```ruby
[0, 2].map do |n|
  next "skip" if n == 0
  n * 2
end
```

Αποτέλεσμα: `["skip", 4]`.

130 / ΛΥΣΗ

Επανάληψη με συνθήκη

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[0, 1, 2]`.

```ruby
index = 0
values = []
while index < 3
  values << index
  index += 1
end
values
```

Η κύρια άσκηση

```ruby
def consume_tokens(tokens, limit)
  result = []
  index = 0
  while index < tokens.length && index < limit
    break if tokens[index] == "END"
    result << tokens[index]
    index += 1
  end
  result
end
```

Το index αυξάνεται μετά την αποθήκευση. Αν ξεχαστεί, η ίδια θέση θα διαβάζεται επ' αόριστον.

Η παραλλαγή

Χρησιμοποίησε until για μέτρηση μέχρι 2 και loop με break για να επιστρέψεις "done".

```ruby
n = 0
until n == 2
  n += 1
end
result = loop do
  break "done"
end
[n, result]
```

Αποτέλεσμα: `[2, "done"]`.

131 / ΛΥΣΗ

Συγκεκριμένα βήματα επανάληψης

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, 3, 4]`.

```ruby
values = []
2.upto(4) { |n| values << n }
values
```

Η κύρια άσκηση

```ruby
def time_marks(start_minute, end_minute, step_size)
  marks = []
  start_minute.step(end_minute, step_size) { |minute| marks << minute }
  marks
end
```

Το end_minute εμφανίζεται μόνο αν το βήμα προσγειώνεται ακριβώς εκεί. Η step δεν προσθέτει αυτόματα μια μικρότερη τελική απόσταση.

Η παραλλαγή

Φτιάξε αντίστροφη μέτρηση από 3 έως 1 και τρεις ετικέτες με times.

```ruby
countdown = []
3.downto(1) { |n| countdown << n }
labels = []
3.times { |index| labels << "N#{index + 1}" }
[countdown, labels]
```

Αποτέλεσμα: `[[3, 2, 1], ["N1", "N2", "N3"]]`.

132 / ΛΥΣΗ

Αρχή και υπόλοιπο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[1, 2], [3]]`.

```ruby
items = [1, 2, 3]
[items.take(2), items.drop(2)]
```

Η κύρια άσκηση

```ruby
def preview_queue(items, count)
  {preview: items.take(count), remaining: items.drop(count)}
end
```

Οι μέθοδοι επιλέγουν κομμάτια χωρίς shift ή pop, επομένως δεν καταναλώνουν τη λίστα.

Η παραλλαγή

Δώσε πρώτο στοιχείο, Array πρώτου στοιχείου και δύο τελευταία.

```ruby
a = [1, 2, 3]
[a.first, a.first(1), a.last(2)]
```

Αποτέλεσμα: `[1, [1], [2, 3]]`.

133 / ΛΥΣΗ

Ομάδες σταθερού μεγέθους

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[1, 2], [3, 4], [5]]`.

```ruby
groups = []
[1, 2, 3, 4, 5].each_slice(2) { |group| groups << group }
groups
```

Η κύρια άσκηση

```ruby
def teams_of_three(names)
  teams = []
  names.each_slice(3) { |team| teams << team }
  teams
end
```

Κάθε στοιχείο εμφανίζεται σε ακριβώς μία ομάδα. Οι επικαλυπτόμενοι γείτονες είναι η επόμενη έννοια.

Η παραλλαγή

Χώρισε τέσσερα ποσά σε ζευγάρια και δώσε ένα άθροισμα ανά ζευγάρι.

```ruby
totals = []
[1, 2, 3, 4].each_slice(2) { |pair| totals << pair.sum }
totals
```

Αποτέλεσμα: `[3, 7]`.

134 / ΛΥΣΗ

Διαδοχικά ζευγάρια

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[1, 4], [4, 9]]`.

```ruby
pairs = []
[1, 4, 9].each_cons(2) { |pair| pairs << pair }
pairs
```

Η κύρια άσκηση

```ruby
def daily_changes(values)
  changes = []
  values.each_cons(2) { |before, after| changes << after - before }
  changes
end
```

Δεν αφαιρούμε την πρώτη τιμή από όλες τις επόμενες. Κάθε αλλαγή αφορά το δικό της γειτονικό ζευγάρι.

Η παραλλαγή

Αντί για διαφορές, δώσε ετικέτες μεταβάσεων στάσεων.

```ruby
routes = []
["A", "B", "C"].each_cons(2) { |a, b| routes << "#{a} → #{b}" }
routes
```

Αποτέλεσμα: `["A → B", "B → C"]`.

135 / ΛΥΣΗ

Συλλογές δίπλα δίπλα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["A", 8], ["B", 9], ["C", nil]]`.

```ruby
["A", "B", "C"].zip([8, 9])
```

Η κύρια άσκηση

```ruby
def name_scores(names, scores)
  names.zip(scores).map { |name, score| {name: name, score: score} }
end
```

Η zip είναι σύνδεση ανά θέση, όχι αναζήτηση κοινού κλειδιού. Το nil δείχνει ότι δεν υπήρχε βαθμός σε εκείνη τη θέση.

Η παραλλαγή

Σύνδεσε τρεις παράλληλες λίστες με μία zip.

```ruby
["A", "B"].zip([1, 2], [true, false])
```

Αποτέλεσμα: `[["A", 1, true], ["B", 2, false]]`.

136 / ΛΥΣΗ

Σελίδες και γειτονικές τιμές · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[2, 5], [9]]`.

```ruby
values = [2, 5, 9]
[values.take(2), values.drop(2)]
```

Η κύρια άσκηση

```ruby
def sequence_summary(values)
  groups = []
  values.each_slice(2) { |pair| groups << pair }
  changes = []
  values.each_cons(2) { |a, b| changes << b - a }
  {preview: values.take(2), groups: groups, changes: changes}
end
```

Η each_slice και η each_cons δεν είναι εναλλάξιμες. Η δεύτερη επαναχρησιμοποιεί στοιχεία σε διαδοχικά παράθυρα.

Η παραλλαγή

Πάρε τις πρώτες τρεις τιμές και δώσε γειτονικά ζευγάρια μόνο αυτών.

```ruby
pairs = []
[1, 2, 3, 4].take(3).each_cons(2) { |pair| pairs << pair }
pairs
```

Αποτέλεσμα: `[[1, 2], [2, 3]]`.

137 / ΛΥΣΗ

Πρόγραμμα ομάδων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva@1", "Jo@2"]`.

```ruby
pairs = ["Eva", "Jo"].zip([1, 2])
pairs.map { |name, seat| "#{name}@#{seat}" }
```

Η κύρια άσκηση

```ruby
def group_schedule(names, seats, meetings)
  groups = []
  names.zip(seats).each_slice(2) { |group| groups << group }
  transitions = []
  meetings.each_cons(2) { |a, b| transitions << "#{a} → #{b}" }
  {groups: groups, transitions: transitions}
end
```

Η zip προηγείται της each_slice, ώστε όνομα και θέση να παραμένουν ενιαία εγγραφή μέσα στην ομάδα.

Η παραλλαγή

Δώσε πλήθος ατόμων σε ήδη έτοιμες ομάδες.

```ruby
[[["A", 1], ["B", 2]], [["C", 3]]].map(&:length)
```

Αποτέλεσμα: `[2, 1]`.

138 / ΛΥΣΗ

Η επανάληψη ως αντικείμενο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[Enumerator, ["[A]", "[B]"]]`.

```ruby
enumerator = ["A", "B"].each
[enumerator.class, enumerator.map { |name| "[#{name}]" }]
```

Η κύρια άσκηση

```ruby
def name_enumerator(names)
  names.each
end
```

Το names.each με block επιστρέφει το Array. Χωρίς block η ίδια κλήση επιστρέφει Enumerator.

Η παραλλαγή

Πάρε Enumerator από δοσμένο αρχείο χωρίς block και μετά καθάρισε τις γραμμές με map.

```ruby
enumerator = File.foreach(__dir__ + "/fixtures/names.txt", encoding: "UTF-8")
enumerator.map(&:chomp)
```

Αποτέλεσμα: `["Eva", "Jo"]`.

139 / ΛΥΣΗ

Ζήτησε το επόμενο στοιχείο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "B"]`.

```ruby
e = ["A", "B"].each
[e.next, e.next]
```

Η κύρια άσκηση

```ruby
def next_record(enumerator)
  {done: false, value: enumerator.next}
  rescue StopIteration
    {done: true, value: nil}
end
```

Η next εδώ είναι μέθοδος Enumerator, διαφορετική από το keyword next που παραλείπει ένα βήμα block.

Η παραλλαγή

Κατανάλωσε μία τιμή και πιάσε την εξάντληση στην επόμενη κλήση.

```ruby
e = [7].each
e.next
begin
  e.next
rescue StopIteration
  "finished"
end
```

Αποτέλεσμα: `"finished"`.

140 / ΛΥΣΗ

Κοίτα χωρίς να προχωρήσεις

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "A", "A", "B"]`.

```ruby
e = ["A", "B"].each
[e.peek, e.peek, e.next, e.next]
```

Η κύρια άσκηση

```ruby
def peek_restart(values)
  e = values.each
  preview = e.peek
  first = e.next
  e.rewind
  [preview, first, e.next]
end
```

Το peek μπορεί να προκαλέσει την παραγωγή της επόμενης τιμής και να την κρατήσει διαθέσιμη, αλλά δεν την καταναλώνει για το επόμενο next.

Η παραλλαγή

Κατανάλωσε και τις δύο τιμές ενός Enumerator, κάνε rewind και ζήτησε ξανά την πρώτη.

```ruby
e = [1, 2].each
e.next
e.next
e.rewind
e.next
```

Αποτέλεσμα: `1`.

141 / ΛΥΣΗ

Προσθήκη θέσης σε μια ακολουθία

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["5: A", "6: B"]`.

```ruby
["A", "B"].map.with_index(5) { |name, number| "#{number}: #{name}" }
```

Η κύρια άσκηση

```ruby
def indexed_labels(names, start = 1)
  names.map.with_index(start) { |name, number| "#{number}. #{name}" }
end
```

Τα δύο στάδια είναι map → Enumerator και with_index → διάσχιση με θέση. Το block συνεχίζει να παράγει τις τιμές της map.

Η παραλλαγή

Δώσε ζευγάρια [τιμή, θέση] από Enumerator με αρχή 10.

```ruby
["A", "B"].each.with_index(10).to_a
```

Αποτέλεσμα: `[["A", 10], ["B", 11]]`.

142 / ΛΥΣΗ

Πρόβλεψη της επόμενης τιμής · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, 4, 4]`.

```ruby
e = [2, 4].each
[e.next, e.peek, e.peek]
```

Η κύρια άσκηση

```ruby
def enumerator_trace(values)
  e = values.each
  trace = [e.peek, e.next, e.peek, e.next]
  e.rewind
  trace << e.next
  trace
end
```

Η ρητή rescue επιτρέπει να φανεί η εξάντληση ως μηχανισμός. Δεν σταματάμε επειδή ένα στοιχείο ήταν falsy.

Η παραλλαγή

Με next και rescue κατανάλωσε πεπερασμένο Enumerator με πραγματικό nil στη μέση.

```ruby
e = [1, nil, 2].each
values = []
begin
  while true
    values << e.next
  end
rescue StopIteration
  values
end
```

Αποτέλεσμα: `[1, nil, 2]`.

143 / ΛΥΣΗ

Έλεγχος μορφής κειμένου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `true`.

```ruby
/\A[A-Z][0-9]\z/.match?("B4")
```

Η κύρια άσκηση

```ruby
def valid_code?(text)
  /\A[A-Z][A-Z][0-9][0-9][0-9]\z/.match?(text)
end
```

Χωρίς τα anchors η regex θα έβρισκε τον κωδικό και μέσα σε μεγαλύτερο κείμενο. Τα ^ και $ έχουν συμπεριφορά ορίων γραμμής.

Η παραλλαγή

Δέξου κωδικό ενός πεζού γράμματος και ενός ψηφίου.

```ruby
/\A[a-z][0-9]\z/.match?("a7")
```

Αποτέλεσμα: `true`.

144 / ΛΥΣΗ

Ομάδες και επαναλήψεις σε regex

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `true`.

```ruby
/\A(?:X|Y)-[0-9]{2}\z/.match?("Y-04")
```

Η κύρια άσκηση

```ruby
def valid_batch_code?(text)
  /\A(?:AB|CD)-[0-9]{1,3}\z/.match?(text)
end
```

Η εναλλακτική περιορίζεται σε ομάδα. Χωρίς ομαδοποίηση τα anchors μπορεί να καλύπτουν διαφορετικούς κλάδους απ’ όσο νομίζεις.

Η παραλλαγή

Δέξου το κυριολεκτικό όνομα file.txt, όχι fileXtxt.

```ruby
[/\Afile\.txt\z/.match?("file.txt"), /\Afile\.txt\z/.match?("fileXtxt")]
```

Αποτέλεσμα: `[true, false]`.

145 / ΛΥΣΗ

Εξαγωγή ονομασμένων πεδίων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["XY:12", "XY", "12"]`.

```ruby
match = /\A(?<code>[A-Z]{2}):(?<count>[0-9]+)\z/.match("XY:12")
[match[0], match[:code], match[:count]]
```

Η κύρια άσκηση

```ruby
def parse_code_quantity(text)
  match = /\A(?<code>[A-Z]{2}):(?<quantity>[0-9]+)\z/.match(text)
  return nil unless match
  {code: match[:code], quantity: Integer(match[:quantity], 10)}
end
```

Πρώτα ελέγχουμε ότι υπάρχει MatchData. Η match? θα έδινε μόνο boolean, χωρίς πρόσβαση στα captures.

Η παραλλαγή

Με αριθμημένα captures χώρισε δύο λέξεις γύρω από /.

```ruby
match = /\A([A-Z]+)\/([A-Z]+)\z/.match("A/BC")
[match[1], match[2]]
```

Αποτέλεσμα: `["A", "BC"]`.

146 / ΛΥΣΗ

Πολλά ταιριάσματα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A12", "B3"]`.

```ruby
"A12 B3".scan(/[A-Z][0-9]+/)
```

Η κύρια άσκηση

```ruby
def template_markers(text)
  text.scan(/\{([a-z_]+)\}/).map { |capture| capture[0] }
end
```

Με ένα capture το scan επιστρέφει π.χ. [["name"]]. Η map ανοίγει το ένα πεδίο που θέλει η σύμβαση.

Η παραλλαγή

Χωρίς captures, βρες όλους τους ακέραιους ως Strings από ένα κείμενο.

```ruby
"room 12, seats 3".scan(/[0-9]+/)
```

Αποτέλεσμα: `["12", "3"]`.

147 / ΛΥΣΗ

Αντικατάσταση με υπολογισμό

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"A4 B6"`.

```ruby
"A2 B3".gsub(/[0-9]+/) { |digits| (Integer(digits, 10) * 2).to_s }
```

Η κύρια άσκηση

```ruby
def render_markers(text, values)
  text.gsub(/\{[a-z_]+\}/) do |token|
    key = token[1...-1]
    values.fetch(key, token)
  end
end
```

Το block παίρνει ολόκληρο token. Η slicing αφαιρεί τα braces και η fetch διατηρεί άγνωστα tokens για να είναι ορατά.

Η παραλλαγή

Με block βάλε αγκύλες γύρω από κάθε αριθμό.

```ruby
"A12 B3".gsub(/[0-9]+/) { |digits| "[#{digits}]" }
```

Αποτέλεσμα: `"A[12] B[3]"`.

148 / ΛΥΣΗ

Κείμενο σε δομημένες τιμές · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Eva"`.

```ruby
/\A(?<name>[A-Za-z]+):(?<count>[0-9]+)\z/.match("Eva:2")[:name]
```

Η κύρια άσκηση

```ruby
def parsed_labels(lines)
  lines.filter_map do |line|
    match = /\A(?<name>[A-Za-z]+):(?<count>[0-9]+)\z/.match(line)
    if match
      "#{match[:name]} (#{Integer(match[:count], 10)})"
    end
  end
end
```

Η μετατροπή Integer γίνεται μόνο αφού έχει επιβεβαιωθεί η μορφή ψηφίων. Δεν χρειάζεται δεύτερη διαδρομή rescue για την ίδια έγκυρη μορφή.

Η παραλλαγή

Αντικατάστησε το σταθερό token {total} με δοσμένο πλήθος μέσω gsub block.

```ruby
total = 3
"Count: {total}".gsub(/\{total\}/) { total.to_s }
```

Αποτέλεσμα: `"Count: 3"`.

149 / ΛΥΣΗ

Χρονικές στιγμές και διάρκειες

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `0.0`.

```ruby
require "time"
a = Time.iso8601("2026-01-10T10:00:00+02:00")
b = Time.iso8601("2026-01-10T08:00:00Z")
a - b
```

Η κύρια άσκηση

```ruby
def meeting_minutes(start_text, finish_text)
  require "time"
  start = Time.iso8601(start_text)
  finish = Time.iso8601(finish_text)
  ((finish - start) / 60).to_i
end
```

Αφαιρούμε χρονικές στιγμές, όχι τις ώρες ως κείμενα. Τα offsets είναι ρητά δεδομένα, οπότε το αποτέλεσμα δεν εξαρτάται από τη ζώνη του μηχανήματος.

Η παραλλαγή

Μετέτρεψε στιγμή σε UTC με getutc και δώσε ISO String.

```ruby
require "time"
Time.iso8601("2026-01-10T10:00:00+02:00").getutc.iso8601
```

Αποτέλεσμα: `"2026-01-10T08:00:00Z"`.

150 / ΛΥΣΗ

Ημερολογιακές ημερομηνίες

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"29/02/2024"`.

```ruby
require "date"
(Date.iso8601("2024-02-28") + 1).strftime("%d/%m/%Y")
```

Η κύρια άσκηση

```ruby
def event_dates(dates)
  require "date"
  dates.map { |text| Date.iso8601(text).strftime("%d/%m/%Y") }
end
```

Η Date λύνει ημερολογιακές μεταβάσεις χωρίς να γράψουμε κανόνες μηνών ή δίσεκτων ετών.

Η παραλλαγή

Πρόσθεσε μία ημέρα στην τελευταία ημέρα του έτους.

```ruby
require "date"
(Date.iso8601("2025-12-31") + 1).iso8601
```

Αποτέλεσμα: `"2026-01-01"`.

151 / ΛΥΣΗ

Σύνολα τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["B"]`.

```ruby
require "set"
(Set.new(["A", "B"]) & Set.new(["B", "C"])).to_a.sort
```

Η κύρια άσκηση

```ruby
def compare_groups(left, right)
  require "set"
  a = Set.new(left)
  b = Set.new(right)
  {all: (a | b).to_a.sort, common: (a & b).to_a.sort, only_left: (a - b).to_a.sort}
end
```

Η διαφορά a - b δεν είναι συμμετρική. Δίνει όσα ανήκουν μόνο στην αριστερή ομάδα.

Η παραλλαγή

Έλεγξε αν το όνομα Eva ανήκει σε Set και αν τα διπλότυπα αυξάνουν το μέγεθος.

```ruby
require "set"
group = Set.new(["Eva", "Eva"])
[group.include?("Eva"), group.size]
```

Αποτέλεσμα: `[true, 1]`.

152 / ΛΥΣΗ

Ακριβείς δεκαδικές τιμές

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"0.3"`.

```ruby
require "bigdecimal"
(BigDecimal("0.1") + BigDecimal("0.2")).to_s("F")
```

Η κύρια άσκηση

```ruby
def decimal_charge(amounts, rate)
  require "bigdecimal"
  multiplier = BigDecimal(rate)
  total = amounts.sum(BigDecimal("0")) { |amount| BigDecimal(amount) * multiplier }
  total.round(2, BigDecimal::ROUND_HALF_UP).to_s("F")
end
```

Η κατασκευή από String προηγείται της αριθμητικής. Η στρογγυλοποίηση κάθε στοιχείου θα μπορούσε να δώσει διαφορετικό τελικό σύνολο.

Η παραλλαγή

Στρογγύλεψε αρνητικό μισό στην ίδια πολιτική HALF_UP.

```ruby
require "bigdecimal"
BigDecimal("-1.005").round(2, BigDecimal::ROUND_HALF_UP).to_s("F")
```

Αποτέλεσμα: `"-1.01"`.

153 / ΛΥΣΗ

Επιλογές εντολής

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[{upper: true}, ["Eva"]]`.

```ruby
require "optparse"
options = {upper: false}
args = ["--upper", "Eva"]
OptionParser.new { |p| p.on("--upper") { options[:upper] = true } }.parse!(args)
[options, args]
```

Η κύρια άσκηση

```ruby
def parse_format_options(args)
  require "optparse"
  options = {upper: false, prefix: ""}
  remaining = args.dup
  parser = OptionParser.new do |p|
    p.on("--upper") { options[:upper] = true }
    p.on("--prefix TEXT") { |text| options[:prefix] = text }
  end
  parser.parse!(remaining)
  {options: options, arguments: remaining}
end
```

Η parse! μεταβάλλει το δοσμένο Array. Το dup χωρίζει τη μεταβολή που χρειάζεται ο parser από τα ορίσματα του καλούντος.

Η παραλλαγή

Χρησιμοποίησε -- για να περάσεις όνομα που αρχίζει με -- ως απλό όρισμα.

```ruby
require "optparse"
args = ["--", "--name"]
OptionParser.new.parse!(args)
args
```

Αποτέλεσμα: `["--name"]`.

154 / ΛΥΣΗ

Εκτέλεση άλλου τοπικού προγράμματος

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Hello Eva\n", "", 0]`.

```ruby
require "open3"
require "rbconfig"
out, err, status = Open3.capture3(RbConfig.ruby, __dir__ + "/support/worker.rb",
"Eva")
[out, err, status.exitstatus]
```

Η κύρια άσκηση

```ruby
def run_local_worker(name)
  require "open3"
  require "rbconfig"
  out, err, status = Open3.capture3(RbConfig.ruby, __dir__ + "/support/worker.rb",
name)
  {output: out, error: err, status: status.exitstatus}
end
```

Το status δεν είναι η έξοδος κειμένου. Το success? ή exitstatus περιγράφει πώς τελείωσε το πρόγραμμα, ενώ stdout και stderr κρατούν τα μηνύματα.

Η παραλλαγή

Με Process.spawn και wait2 εκτέλεσε το ίδιο fixture για Jo, κατευθύνοντας stdout σε tmp/child.txt.

```ruby
require "rbconfig"
path = __dir__ + "/tmp/child.txt"
pid = Process.spawn(RbConfig.ruby, __dir__ + "/support/worker.rb", "Jo", out: path)
finished_pid, status = Process.wait2(pid)
[finished_pid == pid, status.success?, File.read(path)]
```

Αποτέλεσμα: `[true, true, "Hello Jo\n"]`.

155 / ΛΥΣΗ

Μετατροπή τοπικών δεδομένων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"02/03/2026: 2.5"`.

```ruby
require "date"
require "bigdecimal"
"#{Date.iso8601("2026-03-02").strftime("%d/%m/%Y")}:
#{BigDecimal("2.50").to_s("F")}"
```

Η κύρια άσκηση

```ruby
def dated_amount_report(path)
  require "json"
  require "date"
  require "bigdecimal"
  JSON.parse(File.read(path, encoding: "UTF-8")).map do |row|
    date = Date.iso8601(row["date"]).strftime("%d/%m/%Y")
    amount = BigDecimal(row["amount"]).round(2, BigDecimal::ROUND_HALF_UP).to_s("F")
    "#{date}: #{amount}"
  end
end
```

Η JSON.parse δίνει String κλειδιά. Η Date και η BigDecimal αναλαμβάνουν τους κανόνες των δικών τους τύπων.

Η παραλλαγή

Μετάφερε μία δοσμένη Date κατά δύο ημέρες πριν τη μορφοποίηση.

```ruby
require "date"
(Date.iso8601("2026-01-31") + 2).strftime("%d/%m/%Y")
```

Αποτέλεσμα: `"02/02/2026"`.

156 / ΛΥΣΗ

Προστασία από μεταβολή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["A!"], true, false]`.

```ruby
values = ["A"].freeze
values[0] << "!"
[values, values.frozen?, values[0].frozen?]
```

Η κύρια άσκηση

```ruby
def frozen_settings(settings)
  settings.dup.freeze
end
```

Το dup προστατεύει την αρχική εξωτερική δομή από το freeze. Η εργασία ζητά ρηχό αντίγραφο και δεν υπόσχεται ανεξάρτητα nested δεδομένα.

Η παραλλαγή

Πάγωσε ρητά και το εσωτερικό String πριν παγώσεις το Array.

```ruby
values = ["A"]
values.each(&:freeze)
values.freeze
[values.frozen?, values[0].frozen?]
```

Αποτέλεσμα: `[true, true]`.

157 / ΛΥΣΗ

Ισότητα και ταυτότητα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[true, true, false, true, false]`.

```ruby
a = "A"
b = "A"
[a == b, a.eql?(b), a.equal?(b), 1 == 1.0, 1.eql?(1.0)]
```

Η κύρια άσκηση

```ruby
class CodeValue
  attr_reader :code
  def initialize(code)
    @code = code.dup.freeze
  end
  def ==(other)
    other.class == CodeValue && code == other.code
  end
  def eql?(other)
    self == other
  end
  def hash
    code.hash
  end
end
```

Μόνο override της == δεν αρκεί για Hash keys. Η παγωμένη εσωτερική τιμή αποτρέπει μεταβολή του hash μετά την εισαγωγή σε Hash.

Η παραλλαγή

Σύγκρινε 1 και 1.0 ως κλειδιά Hash.

```ruby
{1 => "integer", 1.0 => "float"}.length
```

Αποτέλεσμα: `2`.

158 / ΛΥΣΗ

Μικρή εγγραφή χωρίς πολύ boilerplate

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva", 2]`.

```ruby
DemoRow = Struct.new(:name, :seats, keyword_init: true)
row = DemoRow.new(name: "Eva", seats: 2)
[row.name, row.seats]
```

Η κύρια άσκηση

```ruby
ReportRow = Struct.new(:name, :quantity, :price, keyword_init: true) do
  def total
    quantity * price
  end
end
```

Το Struct αφαιρεί το επαναλαμβανόμενο initialize και attr_accessor. Η total είναι η δική μας συμπεριφορά πάνω στα πεδία.

Η παραλλαγή

Όρισε Struct δύο positional πεδίων και φτιάξε μία παρουσία.

```ruby
PointRow = Struct.new(:x, :y)
p = PointRow.new(2, 3)
[p.x, p.y]
```

Αποτέλεσμα: `[2, 3]`.

159 / ΛΥΣΗ

Τιμή με σταθερά μέλη

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, 3]`.

```ruby
ViewSettings = Data.define(:theme, :size)
original = ViewSettings.new(theme: "light", size: 2)
changed = original.with(size: 3)
[original.size, changed.size]
```

Η κύρια άσκηση

```ruby
DisplaySettings = Data.define(:theme, :tags)

def change_theme(settings, theme)
  settings.with(theme: theme)
end
```

Η Data δίνει σταθερά πεδία, όχι αυτόματα βαθιά immutable δεδομένα. Η with αλλάζει μόνο τα πεδία που δηλώνεις.

Η παραλλαγή

Κράτησε εσωτερικό Array παγωμένο πριν το βάλεις σε Data.

```ruby
FrozenTags = Data.define(:tags)
value = FrozenTags.new(tags: ["a"].freeze)
[value.frozen?, value.tags.frozen?]
```

Αποτέλεσμα: `[true, true]`.

160 / ΛΥΣΗ

Συμπεριφορά πάνω σε ένα αντικείμενο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[A]"`.

```ruby
module DemoFormat
  def bracket(text)
    "[#{text}]"
  end
end
object = Object.new
object.extend(DemoFormat)
object.bracket("A")
```

Η κύρια άσκηση

```ruby
module LabelTools
  def label(name)
    "[#{name}]"
  end
end

def enable_labels(object)
  object.extend(LabelTools)
end
```

Η extend αφορά τον συγκεκριμένο receiver. Δεν ανοίγουμε την Object για να αλλάξουμε όλα τα αντικείμενα.

Η παραλλαγή

Κάνε extend σε κλάση ώστε να αποκτήσει μέθοδο κλάσης.

```ruby
module ClassBanner
  def banner
    "Ready"
  end
end
class BannerHost
  extend ClassBanner
end
BannerHost.banner
```

Αποτέλεσμα: `"Ready"`.

161 / ΛΥΣΗ

Σύντομες μορφές εκφράσεων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"large"`.

```ruby
def size_word(n) = n >= 5 ? "large" : "small"
size_word(5)
```

Η κύρια άσκηση

```ruby
def paid_label(paid) = paid ? "paid" : "pending"

def short_total(quantity, price) = quantity * price
```

Χρησιμοποιούμε αυτή τη μορφή για μικρές σαφείς εκφράσεις. Με πολλούς κανόνες, ο γνωστός κανονικός ορισμός διαβάζεται ευκολότερα.

Η παραλλαγή

Γράψε endless μέθοδο που μορφοποιεί όνομα μέσα σε [].

```ruby
def short_badge(name) = "[#{name}]"
short_badge("Eva")
```

Αποτέλεσμα: `"[Eva]"`.

162 / ΛΥΣΗ

Σύντομες παράμετροι block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[4, 6], ["A", "B"]]`.

```ruby
[[2, 3].map { _1 * 2 }, ["a", "b"].map { it.upcase }]
```

Η κύρια άσκηση

```ruby
def implicit_names(names)
  names.map { _1.strip }.map { it.upcase }
end
```

Και τα δύο blocks έχουν μία απλή ενέργεια. Όταν το όνομα εξηγεί τον ρόλο του στοιχείου, η ρητή παράμετρος παραμένει καλή επιλογή.

Η παραλλαγή

Με _1 κράτησε μόνο τις θετικές ποσότητες.

```ruby
[-1, 0, 2].select { _1 > 0 }
```

Αποτέλεσμα: `[2]`.

163 / ΛΥΣΗ

Σε ποια κλήση ανήκει το block;

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[6, false], [2, true]]`.

RUBY

```ruby
def binding_outer(value)
  [value, block_given?]
end
def binding_inner
  block_given? ? yield(2) : 2
end
a = binding_outer binding_inner { |n| n * 3 }
b = binding_outer binding_inner do |n| n * 3 end
[a, b]
```

Η κύρια άσκηση

RUBY

```ruby
def wrap_values(values)
  "[#{values.join(", ")}]"
end

def wrapped_doubles(numbers)
  wrap_values(numbers.map do |number|
    number * 2
  end)
end
```

Το πρόβλημα είναι πού δένεται το block, όχι αλλαγή στον μηχανισμό της map. Οι παρενθέσεις γύρω από ολόκληρη την εσωτερική κλήση το λύνουν.

Η παραλλαγή

Γράψε ρητά την εσωτερική κλήση με παρενθέσεις σε παράδειγμα δύο μεθόδων.

RUBY

```ruby
def outer_text(value)
  "[#{value}]"
end
def inner_text
  yield("A")
end
outer_text(inner_text { |s| s.downcase })
```

Αποτέλεσμα: `"[a]"`.

164 / ΛΥΣΗ

Προώθηση ορισμάτων και block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"HI EVA"`.

```ruby
def forwarding_target(name, prefix: "")
  text = "#{prefix}#{name}"
  block_given? ? yield(text) : text
end
def forwarding_bridge(...)
  forwarding_target(...)
end
forwarding_bridge("Eva", prefix: "Hi ") { |s| s.upcase }
```

Η κύρια άσκηση

```ruby
def base_format(name, prefix: "")
  text = "#{prefix}#{name}"
  block_given? ? yield(text) : text
end

def relay_all(...)
  base_format(...)
end

def relay_parts(*, **, &)
  base_format(*, **, &)
end
```

Το ... κρατά και το block, όχι μόνο τιμές ορισμάτων. Δεν προσπαθούμε να ξαναχτίσουμε χειροκίνητα την κλήση.

Η παραλλαγή

Προώθησε μόνο block με ανώνυμο & σε μικρή μέθοδο.

```ruby
def block_target
  yield(3)
end
def block_bridge(&)
  block_target(&)
end
block_bridge { |n| n * 2 }
```

Αποτέλεσμα: `6`.

165 / ΛΥΣΗ

Μορφοποίηση τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Seat: 007"`.

```ruby
format("%s: %03d", "Seat", 7)
```

Η κύρια άσκηση

```ruby
def receipt_line(number, amount)
  format("#%03d: %.2f", number, amount)
end
```

Η μορφοποίηση εδώ αφορά παρουσίαση. Ο ακριβής υπολογισμός και η πολιτική στρογγυλοποίησης διδάχτηκαν χωριστά με BigDecimal.

Η παραλλαγή

Χρησιμοποίησε ονομασμένα placeholders για όνομα και δύο θέσεις.

```ruby
format("%<name>s: %<seats>02d", name: "Eva", seats: 2)
```

Αποτέλεσμα: `"Eva: 02"`.

166 / ΛΥΣΗ

Άλλες μορφές literals

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `['Hi #{name}', "Hi Eva", ["tea", "cake"]]`.

```ruby
name = "Eva"
[%q[Hi #{name}], %Q[Hi #{name}], %w[tea cake]]
```

Η κύρια άσκηση

```ruby
def invitation(name)
  <<~TEXT
    Hello #{name}
      Seats ready
  TEXT
end
```

Το <<~ αφαιρεί μόνο την κοινή εσοχή. Η δεύτερη γραμμή έχει δύο επιπλέον κενά που πρέπει να διατηρηθούν.

Η παραλλαγή

Με μονά εισαγωγικά στο heredoc delimiter κράτησε interpolation ως κείμενο.

```ruby
<<~'TEXT'
  Hello #{name}
TEXT
```

Αποτέλεσμα: `"Hello \#{name}\n"`.

167 / ΛΥΣΗ

Χαρακτήρας, byte και encoding

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, 3, [101, 769], 1]`.

```ruby
text = "e\u0301"
[text.length, text.bytesize, text.codepoints, text.grapheme_clusters.length]
```

Η κύρια άσκηση

```ruby
def text_units(text)
  {characters: text.length, bytes: text.bytesize, codepoints: text.codepoints,
graphemes: text.grapheme_clusters.length}
end
```

Το bytesize δεν είναι πλήθος εμφανιζόμενων γραμμάτων. Διάλεξε μονάδα σύμφωνα με την εργασία, όχι με ένα γενικό «μήκος».

Η παραλλαγή

Χώρισε κείμενο a και e με συνδυαστικό τόνο σε grapheme clusters.

```ruby
"ae\u0301".grapheme_clusters
```

Αποτέλεσμα: `["a", "e\u0301"]`.

168 / ΛΥΣΗ

Ταίριασμα δομής πίνακα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Eva: 2"`.

```ruby
case [:book, "Eva", 2]
in [:book, name, seats]
  "#{name}: #{seats}"
else
  nil
end
```

Η κύρια άσκηση

```ruby
def array_command(data)
  case data
  in [:book, name, seats]
    {action: :book, name: name, seats: seats}
  in [:batch, *names]
    {action: :batch, names: names}
  else
    nil
  end
end
```

Τα ονόματα name και seats δεσμεύονται από το pattern. Η ανάθεση αυτή δεν μεταβάλλει το Array εισόδου.

Η παραλλαγή

Με rest πάρε το πρώτο στοιχείο και όλα τα υπόλοιπα ενός μη κενού Array.

```ruby
case [1, 2, 3]
in [first, *rest]
  [first, rest]
end
```

Αποτέλεσμα: `[1, [2, 3]]`.

169 / ΛΥΣΗ

Ταίριασμα δομής εγγραφής

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva", 2, {note: "x"}]`.

```ruby
case {name: "Eva", seats: 2, note: "x"}
in {name:, seats:, **extra}
  [name, seats, extra]
end
```

Η κύρια άσκηση

```ruby
def exact_booking(record)
  case record
  in {name:, seats:, **nil}
    [name, seats]
  else
    nil
  end
end
```

Τα hash patterns δεν ελέγχουν από μόνα τους ότι μια τιμή είναι χρήσιμη ή μη nil. Η κύρια άσκηση αφορά παρουσία και ακριβές σύνολο κλειδιών.

Η παραλλαγή

Με Hash pattern έλεγξε ότι το name είναι String, επιτρέποντας πρόσθετα πεδία.

```ruby
case {name: "Eva", seats: 2}
in {name: String}
  true
else
  false
end
```

Αποτέλεσμα: `true`.

170 / ΛΥΣΗ

Περιορισμοί μέσα στο pattern

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `2`.

```ruby
wanted = "Athens"
case {city: "Athens", seats: 2}
in {city: ^wanted, seats:} if seats > 0
  seats
else
  nil
end
```

Η κύρια άσκηση

```ruby
def seats_in_city(record, city)
  case record
  in {city: ^city, seats:} if seats > 0
    seats
  else
    nil
  end
end
```

Χωρίς ^city, το pattern θα ανέθετε την πόλη της εγγραφής στην τοπική city και δεν θα έλεγχε την επιθυμητή πόλη.

Η παραλλαγή

Δέξου κατάσταση :new ή :queued σε Array δύο στοιχείων και κράτησε το όνομα.

```ruby
case [:queued, "Eva"]
in [:new | :queued, name]
  name
else
  nil
end
```

Αποτέλεσμα: `"Eva"`.

171 / ΛΥΣΗ

Ανάθεση μέσω pattern

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva", 2]`.

```ruby
data = {name: "Eva", seats: 2}
data => {name:, seats:}
[name, seats]
```

Η κύρια άσκηση

```ruby
def required_booking_fields(record)
  record => {name:, seats:}
  [name, seats]
end
```

Η => είναι δήλωση ότι το σχήμα απαιτείται. Η in ταιριάζει όταν θέλεις απλώς ναι ή όχι χωρίς exception.

Η παραλλαγή

Διάλεξε boolean έλεγχο όταν μια ασυμφωνία είναι κανονική πιθανότητα.

```ruby
data = [1]
ok = (data in [a, b])
ok
```

Αποτέλεσμα: `false`.

172 / ΛΥΣΗ

Pattern matching δικών σου αντικειμένων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `5`.

```ruby
class PairPoint
  def deconstruct
    [2, 3]
  end
end
case PairPoint.new
in [x, y]
  x + y
end
```

Η κύρια άσκηση

```ruby
class Position
  def initialize(x, y)
    @x = x
    @y = y
  end
  def deconstruct
    [@x, @y]
  end
  def deconstruct_keys(keys)
    {x: @x, y: @y}
  end
end
```

Τα patterns καλούν πρωτόκολλα, όχι attr_reader αυτόματα. Η deconstruct_keys πρέπει να δέχεται το όρισμα keys ακόμη κι αν δεν το χρησιμοποιεί.

Η παραλλαγή

Όρισε μικρό αντικείμενο που αποσυντίθεται σε Hash με name.

```ruby
class PatternGuest
  def deconstruct_keys(keys)
    {name: "Eva"}
  end
end
case PatternGuest.new
in {name:}
  name
end
```

Αποτέλεσμα: `"Eva"`.

173 / ΛΥΣΗ

Μικρός δρομολογητής μηνυμάτων · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hello Eva"`.

```ruby
case {type: "greet", name: "Eva"}
in {type: "greet", name:}
  "Hello #{name}"
else
  nil
end
```

Η κύρια άσκηση

```ruby
def route_message(message)
  case message
  in {type: "greet", name: String => name}
    "Hello #{name}"
  in {type: "count", values: Array => values} if values.all? { |v| v in Integer }
    values.sum
  else
    nil
  end
end
```

Το `String => name` συνδυάζει έλεγχο τύπου και δέσμευση της ίδιας τιμής. Ο guard ελέγχει και τα στοιχεία του Array πριν από τη sum.

Η παραλλαγή

Με pin δέξου μόνο μήνυμα με δοσμένο id.

```ruby
wanted = 4
case {id: 4, name: "Eva"}
in {id: ^wanted, name:}
  name
else
  nil
end
```

Αποτέλεσμα: `"Eva"`.

174 / ΛΥΣΗ

Δική σου παραγωγή τιμών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["first", "second"]`.

```ruby
e = Enumerator.new do |yielder|
  yielder << "first"
  yielder << "second"
end
[e.next, e.next]
```

Η κύρια άσκηση

```ruby
def label_generator(names)
  Enumerator.new do |yielder|
    names.each { |name| yielder << "Guest #{name}" }
  end
end
```

Η τελική έκφραση του block παραγωγής δεν προστίθεται αυτόματα στην ακολουθία. Μόνο τα yielder << δίνουν στοιχεία.

Η παραλλαγή

Φτιάξε generator που δίνει αριθμό και έπειτα διπλάσιό του για κάθε είσοδο.

```ruby
e = Enumerator.new do |yielder|
  [2, 3].each do |n|
    yielder << n
    yielder << n * 2
  end
end
e.to_a
```

Αποτέλεσμα: `[2, 4, 3, 6]`.

175 / ΛΥΣΗ

Μέθοδος με και χωρίς block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["[A]", "[B]"]`.

```ruby
def each_tag(names)
  return enum_for(__method__, names) unless block_given?
  names.each { |name| yield("[#{name}]") }
end
each_tag(["A", "B"]).to_a
```

Η κύρια άσκηση

```ruby
def each_positive(values)
  return enum_for(__method__, values) unless block_given?
  values.each { |value| yield(value) if value > 0 }
  values
end
```

Ο guard αποτρέπει yield χωρίς block. Περνάμε και values στην enum_for ώστε η μελλοντική κλήση να έχει τα αρχικά ορίσματα.

Η παραλλαγή

Πάρε Enumerator από ήδη υπάρχουσα μέθοδο με to_enum.

```ruby
["A", "B"].to_enum(:each).map(&:downcase)
```

Αποτέλεσμα: `["a", "b"]`.

176 / ΛΥΣΗ

Οι μέθοδοι συλλογών στη δική σου κλάση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `5`.

```ruby
class TwoNumbers
  include Enumerable
  def each
    return enum_for(__method__) unless block_given?
    yield 2
    yield 3
    self
  end
end
TwoNumbers.new.sum
```

Η κύρια άσκηση

```ruby
class NumberShelf
  include Enumerable
  def initialize(values)
    @values = values
  end
  def each
    return enum_for(__method__) unless block_given?
    @values.each { |value| yield(value) }
    self
  end
end
```

Η each είναι το μικρό πρωτόκολλο που συνδέει τη δομή μας με τις έτοιμες πράξεις. Το include μόνο του δεν ξέρει πού βρίσκονται τα στοιχεία.

Η παραλλαγή

Φτιάξε Enumerable που δίνει δύο Strings και χρησιμοποίησε join μέσω to_a.

```ruby
class TwoNames
  include Enumerable
  def each
    return enum_for(__method__) unless block_given?
    yield "Eva"
    yield "Jo"
    self
  end
end
TwoNames.new.to_a.join(", ")
```

Αποτέλεσμα: `"Eva, Jo"`.

177 / ΛΥΣΗ

Υπολογισμός όταν ζητηθεί

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[], [2], [1]]`.

```ruby
seen = []
pipeline = [1, 2, 3].lazy.map { |n| seen << n; n * 2 }
before = seen.dup
result = pipeline.take(1).force
[before, result, seen]
```

Η κύρια άσκηση

```ruby
def lazy_labels(names, audit)
  names.lazy.map do |name|
    audit << name
    name.upcase
  end
end
```

Το audit επιτρέπεται να αλλάζει μόνο όταν εκτελείται το block. Η lazy δημιουργία δεν είναι από μόνη της εκτέλεση.

Η παραλλαγή

Υλοποίησε όλη μια πεπερασμένη lazy ακολουθία με to_a.

```ruby
[1, 2, 3].lazy.map { |n| n + 1 }.to_a
```

Αποτέλεσμα: `[2, 3, 4]`.

178 / ΛΥΣΗ

Σταμάτημα με αρκετά αποτελέσματα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[3, 4], [1, 2, 3, 4]]`.

```ruby
seen = []
result = [1, 2, 3, 4, 5].lazy.map { |n| seen << n; n }.select { |n| n >= 3
}.take(2).force
[result, seen]
```

Η κύρια άσκηση

```ruby
def first_matching(values, minimum, count, audit)
  values.lazy.map { |value| audit << value; value }.select { |value| value >=
minimum }.take(count).force
end
```

Το audit μετρά είσοδο και το take έξοδο. Μεταφορά του take πριν από το φίλτρο θα άλλαζε τη σύμβαση.

Η παραλλαγή

Πάρε τα πρώτα δύο άρτια από πεπερασμένο lazy Range.

```ruby
(1..10).lazy.select { |n| n % 2 == 0 }.take(2).force
```

Αποτέλεσμα: `[2, 4]`.

179 / ΛΥΣΗ

Ακολουθία χωρίς έτοιμο τέλος

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[6, 8, 10]`.

```ruby
(3..).lazy.map { |n| n * 2 }.take(3).force
```

Η κύρια άσκηση

```ruby
def next_ticket_numbers(start, count)
  (start..).lazy.take(count).force
end
```

Το take μπαίνει πριν από force. Ένα φίλτρο που δεν βρίσκει ποτέ αρκετές τιμές δεν εγγυάται τέλος, ακόμη κι αν υπάρχει take.

Η παραλλαγή

Από θετικούς ακεραίους πάρε τα πρώτα τρία πολλαπλάσια του 5.

```ruby
(1..).lazy.select { |n| n % 5 == 0 }.take(3).force
```

Αποτέλεσμα: `[5, 10, 15]`.

180 / ΛΥΣΗ

Σύνδεση και επανάληψη ακολουθιών

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "B", "C"]`.

```ruby
["A"].chain(["B", "C"]).to_a
```

Η κύρια άσκηση

```ruby
def repeated_roster(names, rounds)
  names.cycle(rounds).to_a
end
```

Η κύρια άσκηση έχει ρητά πεπερασμένους γύρους. Η παραλλαγή περιορίζει τον χωρίς όριο κύκλο πριν από την υλοποίηση.

Η παραλλαγή

Σύνδεσε μία εισαγωγική τιμή με κύκλο A/B και πάρε μόνο πέντε τιμές.

```ruby
["start"].chain(["A", "B"].cycle).lazy.take(5).force
```

Αποτέλεσμα: `["start", "A", "B", "A", "B"]`.

181 / ΛΥΣΗ

Ομάδες γειτονικών στοιχείων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["a", ["a", "a"]], ["b", ["b"]], ["a", ["a"]]]`.

```ruby
["a", "a", "b", "a"].chunk { |value| value }.to_a
```

Η κύρια άσκηση

```ruby
def consecutive_runs(values)
  values.chunk_while { |before, after| after == before + 1 }.to_a
end
```

Η γειτνίαση είναι μέρος του ζητούμενου. Η sort θα άλλαζε τις ομάδες. Στη chunk το nil και ορισμένα ειδικά Symbols έχουν πρόσθετη σημασία, γι' αυτό εδώ χρησιμοποιούμε String κλειδιά στο παράδειγμα.

Η παραλλαγή

Σύγκρινε πλήθος ομάδων chunk και group_by σε a,a,b,a.

```ruby
values = ["a", "a", "b", "a"]
[values.chunk { |v| v }.to_a.length, values.group_by { |v| v }.length]
```

Αποτέλεσμα: `[3, 2]`.

182 / ΛΥΣΗ

Όσο ισχύει ο κανόνας στην αρχή

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[1, 2], [5, 1]]`.

```ruby
values = [1, 2, 5, 1]
[values.take_while { |n| n < 3 }, values.drop_while { |n| n < 3 }]
```

Η κύρια άσκηση

```ruby
def leading_small(values, limit)
  [values.take_while { |value| value < limit }, values.drop_while { |value| value < limit }]
end
```

Η select θα κρατούσε και το τελευταίο 1 του παραδείγματος. Η take_while αφορά prefix, όχι όλες τις αντιστοιχίσεις.

Η παραλλαγή

Αφαίρεσε μόνο τα αρχικά κενά Strings, κρατώντας μεταγενέστερα κενά.

```ruby
["", "", "A", "", "B"].drop_while(&:empty?)
```

Αποτέλεσμα: `["A", "", "B"]`.

183 / ΛΥΣΗ

Γραμμές και στήλες

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[["A", "B"], [2, 3]]`.

```ruby
[["A", 2], ["B", 3]].transpose
```

Η κύρια άσκηση

```ruby
def table_columns(rows)
  rows.transpose
end
```

Η transpose αλλάζει το σχήμα. Η zip έχει διαφορετική σύμβαση για λίστες διαφορετικού μήκους.

Η παραλλαγή

Κάνε transpose δύο φορές σε ορθογώνιο πίνακα.

```ruby
[[1, 2], [3, 4]].transpose.transpose
```

Αποτέλεσμα: `[[1, 2], [3, 4]]`.

184 / ΛΥΣΗ

Περιστροφή σειράς

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["B", "C", "A"]`.

```ruby
["A", "B", "C"].rotate(1)
```

Η κύρια άσκηση

```ruby
def rotated_roster(names, offset)
  names.rotate(offset)
end
```

Δεν χρειάζεται χειροκίνητος υπολογισμός modulo. Η rotate ήδη χειρίζεται τα όρια της συλλογής.

Η παραλλαγή

Με reverse_each φτιάξε αντίστροφες ετικέτες χωρίς reverse.

```ruby
["A", "B", "C"].reverse_each.map { |name| "[#{name}]" }
```

Αποτέλεσμα: `["[C]", "[B]", "[A]"]`.

185 / ΛΥΣΗ

Ουρά που διαβάζεται σταδιακά · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[2, 3]`.

```ruby
source = Enumerator.new { |y| ["2", "x", "3"].each { |s| y << s } }
source.lazy.filter_map { |s| Integer(s, 10, exception: false) }.take(2).force
```

Η κύρια άσκηση

```ruby
def first_positive_jobs(lines, count, audit)
  source = Enumerator.new do |yielder|
    lines.each do |line|
      audit << line
      yielder << line
    end
  end
  source.lazy.filter_map { |line| Integer(line, 10, exception: false) }.select { |n| n > 0 }.take(count).force
end
```

Δεν μετατρέπουμε τον generator σε Array πριν από το lazy. Αν το κάναμε, θα είχε ήδη εξεταστεί όλη η πηγή.

Η παραλλαγή

Χρησιμοποίησε lazy ακολουθία για τις πρώτες δύο μη κενές καθαρές γραμμές.

```ruby
[" ", " A ", "B", "C"].lazy.map(&:strip).reject(&:empty?).take(2).force
```

Αποτέλεσμα: `["A", "B"]`.

186 / ΛΥΣΗ

Δικά σου συγκρίσιμα αντικείμενα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[1, -1]`.

```ruby
[[1, "B"] <=> [1, "A"], [1, "A"] <=> [2, "A"]]
```

Η κύρια άσκηση

```ruby
class PriorityItem
  include Comparable
  attr_reader :priority, :name
  def initialize(priority, name)
    @priority = priority
    @name = name
  end
  def <=>(other)
    return nil unless other.class == PriorityItem
    [priority, name] <=> [other.priority, other.name]
  end
end
```

Η <=> επιστρέφει αποτέλεσμα σύγκρισης, όχι boolean. Η sort χρησιμοποιεί το πρωτόκολλο της κλάσης.

Η παραλλαγή

Παρατήρησε το πρόσημο της σύγκρισης ακεραίων και την ισότητα.

```ruby
[2 <=> 7, 7 <=> 2, 2 <=> 2]
```

Αποτέλεσμα: `[-1, 1, 0]`.

187 / ΛΥΣΗ

Τελεστές ως μέθοδοι

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"value for name"`.

```ruby
class SimpleLookup
  def [](key)
    "value for #{key}"
  end
end
SimpleLookup.new[:name]
```

Η κύρια άσκηση

```ruby
class SettingsBox
  attr_reader :values
  def initialize(values)
    @values = values.dup
  end
  def [](key)
    @values[key]
  end
  def []=(key, value)
    @values[key] = value
  end
  def +(other)
    SettingsBox.new(values.merge(other.values))
  end
end
```

Η + παράγει νέο αντικείμενο όπως ορίζει η σύμβαση. Ο setter αντίθετα επιτρέπεται να μεταβάλει το δικό του Hash.

Η παραλλαγή

Όρισε + για αντικείμενο που κρατά ακέραιο πλήθος.

```ruby
class CountValue
  attr_reader :value
  def initialize(value)
    @value = value
  end
  def +(other)
    CountValue.new(value + other.value)
  end
end
(CountValue.new(2) + CountValue.new(3)).value
```

Αποτέλεσμα: `5`.

188 / ΛΥΣΗ

Μέθοδος σε ένα μόνο αντικείμενο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[Eva]"`.

```ruby
special = "Eva"
def special.badge
  "[#{self}]"
end
special.badge
```

Η κύρια άσκηση

```ruby
class LabelFactory
  class << self
    def special(name)
      object = name.dup
      def object.badge
        "VIP #{self}"
      end
      object
    end
  end
end
```

Η badge βρίσκεται στο ένα String και δεν προστίθεται στην String κλάση. Το self μέσα στη badge είναι εκείνο το String.

Η παραλλαγή

Δώσε δύο class methods μέσω class << self.

```ruby
class StatusWords
  class << self
    def ready
      "ready"
    end
    def pending
      "pending"
    end
  end
end
[StatusWords.ready, StatusWords.pending]
```

Αποτέλεσμα: `["ready", "pending"]`.

189 / ΛΥΣΗ

Η σειρά αναζήτησης μεθόδων

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["[A]", [BeforeText, PlainText]]`.

RUBY

```ruby
module BeforeText
  def text
    "[#{super}]"
  end
end
class PlainText
  prepend BeforeText
  def text
    "A"
  end
end
[PlainText.new.text, PlainText.ancestors.first(2)]
```

Η κύρια άσκηση

RUBY

```ruby
module FormatAudit
  def format(name)
    @audit << "before"
    result = super(name)
    @audit << "after"
    result
  end
end
class CoreFormatter
  prepend FormatAudit
  def initialize(audit)
    @audit = audit
  end
  def format(name)
    @audit << "core"
    name.upcase
  end
end
```

Το super δεν σημαίνει πάντα «γονική κλάση». Σημαίνει την επόμενη υλοποίηση στην αλυσίδα αναζήτησης.

Η παραλλαγή

Σύγκρινε τη θέση module που γίνεται include αντί για prepend.

```ruby
module IncludedText
end
class IncludingText
  include IncludedText
end
IncludingText.ancestors.first(2)
```

Αποτέλεσμα: `[IncludingText, IncludedText]`.

190 / ΛΥΣΗ

Επιλογή μεθόδου με όνομα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[true, "EVA"]`.

```ruby
text = "Eva"
[text.respond_to?(:upcase), text.public_send(:upcase)]
```

Η κύρια άσκηση

```ruby
def selected_case(text, style)
  unless [:upcase, :downcase].include?(style) && text.respond_to?(style)
    raise ArgumentError, "unknown style"
  end
  text.public_send(style)
end
```

Η respond_to? από μόνη της δεν ορίζει τις επιλογές της εφαρμογής. Η ρητή λίστα εκφράζει ποιες μορφές υποστηρίζει αυτός ο formatter.

Η παραλλαγή

Πάρε Method από το ίδιο επιλεγμένο όνομα και κάλεσέ το.

```ruby
name = :upcase
"Jo".method(name).call
```

Αποτέλεσμα: `"JO"`.

191 / ΛΥΣΗ

Ορισμός μεθόδων από δεδομένα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[true, false]`.

```ruby
class ColourFlags
  [:red, :blue].each do |colour|
    define_method("#{colour}?") { colour == :red }
  end
end
[ColourFlags.new.red?, ColourFlags.new.blue?]
```

Η κύρια άσκηση

```ruby
class StatusFlags
  def initialize(status)
    @status = status
  end
  [:queued, :done, :cancelled].each do |state|
    define_method("#{state}?") { @status == state }
  end
end
```

Το state είναι η τιμή που θυμάται το block της συγκεκριμένης δήλωσης. Το @status διαβάζεται από το instance την ώρα της κλήσης.

Η παραλλαγή

Δημιούργησε μέθοδο που δέχεται ένα όρισμα μέσω παραμέτρου block.

```ruby
class DynamicGreeting
  define_method(:hello) { |name| "Hi #{name}" }
end
DynamicGreeting.new.hello("Eva")
```

Αποτέλεσμα: `"Hi Eva"`.

192 / ΛΥΣΗ

Μικρές τιμές και δυναμικές κλήσεις · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[[1, "A"], [1, "B"], [2, "A"]]`.

RUBY
```ruby
values = [[2, "A"], [1, "B"], [1, "A"]]
values.sort
```

Η κύρια άσκηση

RUBY
```ruby
class RankedCode
  include Comparable
  attr_reader :rank, :code
  def initialize(rank, code)
    @rank = rank
    @code = code.dup.freeze
  end
  def <=>(other)
    return nil unless other.class == RankedCode
    [rank, code] <=> [other.rank, other.code]
  end
  def eql?(other)
    other.class == RankedCode && self == other
  end
  def hash
    [rank, code].hash
  end
  def plain
    code
  end
  def upper
    code.upcase
  end
  def format(style)
    raise ArgumentError, "unknown style" unless [:plain, :upper].include?(style)
    public_send(style)
  end
end
```

Η σειρά και η ισότητα χρησιμοποιούν τα ίδια δύο πεδία. Η μορφοποίηση δεν αλλάζει την τιμή που λειτουργεί ως κλειδί.

Η παραλλαγή

Σύγκρινε και αποδιπλοποίησε δύο ίσα έτοιμα Array κλειδιά.

```ruby
[[1, "a"], [1, "a"], [2, "a"]].uniq.sort
```

Αποτέλεσμα: `[[1, "a"], [2, "a"]]`.

193 / ΛΥΣΗ

Αντιγραφή και αρχικοποίηση αντιγράφου

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[false, true]`.

```ruby
original = ["A"].freeze
[original.dup.frozen?, original.clone.frozen?]
```

Η κύρια άσκηση

```ruby
class TaggedEntry
  attr_reader :name, :tags
  def initialize(name, tags)
    @name = name
    @tags = tags.dup
  end
  def initialize_copy(other)
    super
    @tags = other.tags.dup
  end
end
```

Η αντιγραφή των εσωτερικών Strings δεν ζητείται εδώ. Η initialize_copy ορίζει ρητά το επιθυμητό βάθος αντί να υποσχόμαστε γενικό deep copy.

Η παραλλαγή

Παρατήρησε ότι clone κρατά singleton method ενώ dup όχι.

```ruby
a = "A"
def a.badge
  "VIP"
end
[a.clone.badge, a.dup.respond_to?(:badge)]
```

Αποτέλεσμα: `["VIP", false]`.

194 / ΛΥΣΗ

Ορισμός εναλλακτικού ονόματος

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"A"`.

```ruby
class AliasDemo
  def render
    "A"
  end
  alias_method :text, :render
end
AliasDemo.new.text
```

Η κύρια άσκηση

```ruby
class LegacyLabel
  def initialize(name)
    @name = name
  end
  def render
    "[#{@name}]"
  end
  alias_method :text, :render
  alias title render
end
```

Το alias αποθηκεύει δεύτερη πρόσβαση στην τότε υλοποίηση, δεν είναι δυναμική προώθηση προς οποιονδήποτε μελλοντικό ορισμό του αρχικού ονόματος.

Η παραλλαγή

Σε δύο υποκλάσεις σύγκρινε remove_method και undef_method πάνω στη label.

RUBY

```ruby
class RemovalBase
  def label
    "base"
  end
end
class RemovedLabel < RemovalBase
  def label
    "child"
  end
  remove_method :label
end
class UndefinedLabel < RemovalBase
  undef_method :label
end
[RemovedLabel.new.label, UndefinedLabel.new.respond_to?(:label)]
```

Αποτέλεσμα: `["base", false]`.

195 / ΛΥΣΗ

Απουσία μεθόδου ως πρωτόκολλο

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Eva", true]`.

```ruby
class MissingDemo
  def method_missing(name, *args, &block)
    return "Eva" if name == :name && args.empty? && !block
    super
  end
  def respond_to_missing?(name, include_private = false)
    name == :name || super
  end
end
[MissingDemo.new.name, MissingDemo.new.respond_to?(:name)]
```

Η κύρια άσκηση

```ruby
class DynamicRecord
  def initialize(fields)
    @fields = fields.slice(:name, :seats)
  end
  def method_missing(name, *args, &block)
    if @fields.key?(name) && args.empty? && !block
      @fields[name]
    else
      super
    end
  end
  def respond_to_missing?(name, include_private = false)
    @fields.key?(name) || super
  end
end
```

Δεν επιστρέφουμε nil για κάθε άγνωστη κλήση. Αυτό θα έκρυβε τυπογραφικά λάθη και θα διαφωνούσε με το δημόσιο πρωτόκολλο του αντικειμένου.

Η παραλλαγή

Δείξε ότι ένα άγνωστο όνομα παραμένει NoMethodError όταν γίνεται super.

```ruby
class EmptyDynamic
  def method_missing(name, *args, &block)
    super
  end
end
begin
  EmptyDynamic.new.unknown
rescue NoMethodError
  "unknown method"
end
```

Αποτέλεσμα: `"unknown method"`.

196 / ΛΥΣΗ

Διαφορετικό self μέσα στο block

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"Hi EVA"`.

```ruby
"Eva".instance_exec("Hi") { |prefix| "#{prefix} #{upcase}" }
```

Η κύρια άσκηση

```ruby
def apply_object_rule(object, value, &rule)
  object.instance_exec(value, &rule)
end
```

Το instance_exec αλλάζει receiver για μεθόδους και @μεταβλητές, όχι τον δεσμό ήδη υπαρχουσών τοπικών μεταβλητών του closure.

Η παραλλαγή

Με class_exec πρόσθεσε reader/writer από όνομα πεδίου σε ανώνυμη κλάση.

```ruby
klass = Class.new
klass.class_exec(:name) { |field| attr_accessor field }
object = klass.new
object.name = "Eva"
object.name
```

Αποτέλεσμα: `"Eva"`.

197 / ΛΥΣΗ

Μικρή εσωτερική γλώσσα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["Name", "Seats"]`.

```ruby
class TinyDefinition
  attr_reader :names
  def initialize
    @names = []
  end
  def column(name)
    @names << name
  end
end
d = TinyDefinition.new
d.instance_exec { column("Name"); column("Seats") }
d.names
```

Η κύρια άσκηση

```ruby
class ReportDefinition
  def self.build(&block)
    definition = new
    definition.instance_exec(&block)
    definition
  end
  def initialize
    @columns = []
  end
  def column(title, &formatter)
    @columns << [title, formatter]
    self
  end
  def render(record)
    @columns.map { |title, formatter| [title, formatter.call(record)] }
  end
end
```

Το DSL παραμένει Ruby κώδικας: δεν κάνει parsing ή eval κειμένου. Η instance_exec χρησιμεύει μόνο για τη σύντομη μορφή δηλώσεων.

Η παραλλαγή

Χτίσε μία μικρή λίστα τίτλων με block που δέχεται ρητό receiver αντί να αλλάζει self.

```ruby
def collect_titles
  titles = []
  yield(titles)
  titles
end
collect_titles { |titles| titles << "Name"; titles << "Seats" }
```

Αποτέλεσμα: `["Name", "Seats"]`.

198 / ΛΥΣΗ

Μικρό API αναφορών · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"[Eva]"`.

```ruby
klass = Class.new
klass.define_method(:label) { |record| "[#{record[:name]}]" }
klass.new.label({name: "Eva"})
```

Η κύρια άσκηση

```ruby
def column_object(formatters)
  allowed = [:name, :seats, :label]
  raise ArgumentError, "unknown column" unless formatters.keys.all? { |name|
allowed.include?(name) }
  klass = Class.new
  formatters.each do |name, formatter|
    klass.define_method(name) { |record| formatter.call(record) }
  end
  klass.new
end
```

Η νέα κλάση ανήκει στη συγκεκριμένη κλήση της factory. Δεν αλλάζουμε μία κοινή κλάση κάθε φορά που ζητείται διαφορετική αναφορά.

Η παραλλαγή

Πρόσθεσε δύο δυναμικές constant-return μεθόδους σε νέα κλάση.

```ruby
klass = Class.new
{left: "L", right: "R"}.each { |name, value| klass.define_method(name) { value } }
object = klass.new
[object.left, object.right]
```

Αποτέλεσμα: `["L", "R"]`.

199 / ΛΥΣΗ

Αλλαγή συμπεριφοράς σε περιορισμένο scope

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["plain", "[plain]"]`.

RUBY

```ruby
class DemoRaw
  def text
    "plain"
  end
end
module DemoRefinement
  refine DemoRaw do
    def text
      "[#{super}]"
    end
  end
end
module DemoRefinedScope
  using DemoRefinement
  def self.render(object)
    object.text
  end
end
[DemoRaw.new.text, DemoRefinedScope.render(DemoRaw.new)]
```

Η κύρια άσκηση

RUBY

```ruby
class RawLabel
  def text
    "plain"
  end
end
module DecoratedLabel
  refine RawLabel do
    def text
      "[#{super}]"
    end
  end
end
module FancyScope
  using DecoratedLabel
  def self.render(label)
    label.text
  end
end
```

Το ίδιο αντικείμενο δίνει διαφορετική συμπεριφορά ανάλογα με το λεκτικό σημείο της κλήσης. Το using ενεργοποιείται πριν από τον ορισμό της render σε εκείνο το module.

Η παραλλαγή

Όρισε refinement μιας δικής σου κλάσης που κάνει κεφαλαίο κείμενο μόνο σε ένα module.

```ruby
class ScopeWord
  def text
    "hello"
  end
end
module UpperWord
  refine ScopeWord do
    def text
      super.upcase
    end
  end
end
module WordScope
  using UpperWord
  def self.read
    ScopeWord.new.text
  end
end
[WordScope.read, ScopeWord.new.text]
```

Αποτέλεσμα: `["HELLO", "hello"]`.

200 / ΛΥΣΗ

Περιορισμένη επανάληψη αποτυχημένης προσπάθειας

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `2`.

```ruby
attempts = 0
begin
  attempts += 1
  raise ArgumentError, "again" if attempts < 2
  attempts
rescue ArgumentError
  retry if attempts < 2
  raise
end
```

Η κύρια άσκηση

```ruby
def retry_local(max_attempts, &action)
  attempts = 0
  begin
    attempts += 1
    action.call
  rescue ArgumentError
    retry if attempts < max_attempts
    raise
  end
end
```

Η επιτυχία ορίζεται από την απουσία εξαίρεσης, όχι από truthiness του αποτελέσματος. Το false δεν αποτελεί λόγο νέας προσπάθειας.

Η παραλλαγή

Κάνε τοπικό retry μέχρι τρίτη προσπάθεια και κράτησε το ίχνος των προσπαθειών.

```ruby
events = []
begin
  events << "try"
  raise ArgumentError if events.length < 3
rescue ArgumentError
  retry if events.length < 3
end
events
```

Αποτέλεσμα: `["try", "try", "try"]`.

201 / ΛΥΣΗ

Δύο ανεξάρτητες εργασίες

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["A", "B"]`.

```ruby
threads = ["a", "b"].map { |name| Thread.new { name.upcase } }
threads.each(&:join)
threads.map(&:value)
```

Η κύρια άσκηση

```ruby
def threaded_labels(names)
  threads = names.map { |name| Thread.new { "[#{name.upcase}]" } }
  threads.each(&:join)
  threads.map(&:value)
end
```

Δεν βασιζόμαστε στη σειρά που τερμάτισαν τα threads. Το Array threads κρατά τη σειρά που ορίζει η σύμβαση.

Η παραλλαγή

Ξεκίνα δύο διαφορετικές ανεξάρτητες εργασίες και συγκέντρωσε με καθορισμένη σειρά.

```ruby
first = Thread.new { "A".downcase }
second = Thread.new { [2, 3].sum }
first.join
second.join
[first.value, second.value]
```

Αποτέλεσμα: `["a", 5]`.

202 / ΛΥΣΗ

Κοινή μεταβαλλόμενη κατάσταση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `6`.

```ruby
counter = 0
mutex = Mutex.new
threads = Array.new(2) { Thread.new { 3.times { mutex.synchronize { counter += 1 } } } }
threads.each(&:join)
counter
```

Η κύρια άσκηση

```ruby
def synchronized_count(workers, per_worker)
  counter = 0
  mutex = Mutex.new
  threads = Array.new(workers) do
    Thread.new do
      per_worker.times { mutex.synchronize { counter += 1 } }
    end
  end
  threads.each(&:join)
  counter
end
```

Η mutex πρέπει να είναι η ίδια για όλα τα threads. Το ότι ένας μη προστατευμένος δοκιμαστικός κώδικας μπορεί να περάσει μία φορά δεν αποδεικνύει ορθότητα.

Η παραλλαγή

Με την ίδια mutex προστάτευσε προσθήκες δύο threads σε Hash μετρητή.

```ruby
counts = Hash.new(0)
mutex = Mutex.new
threads = Array.new(2) { Thread.new { 3.times { mutex.synchronize { counts[:done] += 1 } } } }
threads.each(&:join)
counts
```

Αποτέλεσμα: `{done: 6}`.

203 / ΛΥΣΗ

Επικοινωνία μεταξύ threads

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `"EVA"`.

```ruby
queue = Queue.new
worker = Thread.new do
  name = queue.pop
  name.upcase
end
queue << "Eva"
worker.join
worker.value
```

Η κύρια άσκηση

```ruby
def queued_labels(names)
  queue = Queue.new
  worker = Thread.new do
    result = []
    loop do
      name = queue.pop
      break if name.nil?
      result << name.upcase
    end
    result
  end
  names.each { |name| queue << name }
  queue << nil
  worker.join
  worker.value
end
```

Το σήμα τέλος τοποθετείται μετά τις εργασίες. Χωρίς αυτό ο worker θα περίμενε για πάντα στην επόμενη pop.

Η παραλλαγή

Στείλε δύο ακεραίους και ρητό :done ως σήμα, με άθροισμα στον καταναλωτή.

```ruby
queue = Queue.new
worker = Thread.new do
  total = 0
  loop do
    value = queue.pop
    break if value == :done
    total += value
  end
  total
end
queue << 2
queue << 3
queue << :done
worker.join
worker.value
```

Αποτέλεσμα: `5`.

204 / ΛΥΣΗ

Μικρή ουρά εργασιών με έλεγχο · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `[{id: 1, name: "A"}, {id: 2, name: "B"}]`.

```ruby
queue = Queue.new
queue << {id: 2, name: "B"}
queue << {id: 1, name: "A"}
[queue.pop, queue.pop].sort_by { |row| row[:id] }
```

Η κύρια άσκηση

```ruby
def process_jobs(jobs)
  queue = Queue.new
  results = []
  mutex = Mutex.new
  workers = Array.new(2) do
    Thread.new do
      loop do
        job = queue.pop
        break if job.nil?
        row = [job[:id], job[:name].upcase]
        mutex.synchronize { results << row }
      end
    end
  end
  jobs.each { |job| queue << job }
  2.times { queue << nil }
  workers.each(&:join)
  results.sort_by { |row| row[0] }
end
```

Τα δύο nil τερματίζουν από έναν worker το καθένα. Η τελική ταξινόμηση ορίζει σειρά ανεξάρτητη από τον scheduler.

Η παραλλαγή

Συγκέντρωσε την τιμή επιστροφής δύο threads χωρίς κοινό Array που μεταβάλλεται από αυτά.

```ruby
workers = [1, 2].map { |n| Thread.new { n * 10 } }
workers.each(&:join)
workers.map(&:value)
```

Αποτέλεσμα: `[10, 20]`.

205 / ΛΥΣΗ

Έξοδος από εμφωλευμένα blocks

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `3`.

```ruby
catch(:found) do
  [[1, 2], [3, 4]].each do |group|
    group.each { |n| throw(:found, n) if n > 2 }
  end
  nil
end
```

Η κύρια άσκηση

```ruby
def marker_position(groups, marker)
  catch(:marker_found) do
    groups.each_with_index do |group, group_index|
      group.each_with_index do |value, index|
        throw(:marker_found, [group_index, index]) if value == marker
      end
    end
    nil
  end
end
```

Ένα break στο εσωτερικό each θα σταματούσε μόνο εκείνο το loop. Το throw δίνει αποτέλεσμα σε όλο το catch.

Η παραλλαγή

Χρησιμοποίησε throw για προκαθορισμένη πρόωρη έξοδο με String αποτέλεσμα.

```ruby
catch(:finish) do
  [1, 2, 3].each { |n| throw(:finish, "done") if n == 2 }
  "not found"
end
```

Αποτέλεσμα: `"done"`.

206 / ΛΥΣΗ

Παύση και συνέχιση εργασίας

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["first", "last", false]`.

```ruby
fiber = Fiber.new do
  Fiber.yield("first")
  "last"
end
[fiber.resume, fiber.resume, fiber.alive?]
```

Η κύρια άσκηση

```ruby
def label_fiber(name)
  Fiber.new do
    Fiber.yield("prepare #{name}")
    Fiber.yield("ready #{name}")
    "done"
  end
end
```

Η Fiber.yield δεν είναι το yield προς block μιας μεθόδου. Εδώ η εναλλαγή ελέγχεται ρητά από resume και δεν προϋποθέτει δεύτερο thread.

Η παραλλαγή

Πέρασε όνομα στη δεύτερη resume και χρησιμοποίησέ το ως επιστροφή της Fiber.yield.

```ruby
fiber = Fiber.new do
  name = Fiber.yield("name?")
  "Hi #{name}"
end
[fiber.resume, fiber.resume("Eva")]
```

Αποτέλεσμα: `["name?", "Hi Eva"]`.

207 / ΛΥΣΗ

Πού συνεχίζει η εκτέλεση; · Εμπέδωση

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `["pause", "finish", 7]`.

```ruby
fiber = Fiber.new { Fiber.yield("pause"); "finish" }
thread = Thread.new { 7 }
thread.join
[fiber.resume, fiber.resume, thread.value]
```

Η κύρια άσκηση

```ruby
def execution_trace(failures)
  attempts = 0
  begin
    attempts += 1
    raise ArgumentError if attempts <= failures
  rescue ArgumentError
    retry if attempts <= failures
    raise
  end
  caught = catch(:done) { throw(:done, "caught") }
  fiber = Fiber.new { Fiber.yield("pause"); "finish" }
  worker = Thread.new { "thread" }
  worker.join
  {attempts: attempts, caught: caught, fiber: [fiber.resume, fiber.resume], thread:
> worker.value}
end
```

Η καθορισμένη σειρά συλλογής κάνει το αποτέλεσμα σταθερό. Η άσκηση ελέγχει πού συνεχίζει ο κώδικας, όχι πόσο γρήγορα τρέχει.

Η παραλλαγή

Χρησιμοποίησε catch για να πάρεις την πρώτη τιμή που δίνει μια Fiber.

```ruby
fiber = Fiber.new { Fiber.yield("ready"); "done" }
catch(:result) { throw(:result, fiber.resume) }
```

Αποτέλεσμα: `"ready"`.

208 / ΛΥΣΗ

Απομονωμένες εργασίες με μηνύματα

Το μικρό παράδειγμα

Η τελευταία έκφραση δίνει `6`.

```ruby
worker = Ractor.new do
  number = Ractor.receive
  number * 2
end
worker.send(3)
worker.take
```

Η κύρια άσκηση

```ruby
def ractor_doubles(numbers)
  worker = Ractor.new do
    values = Ractor.receive
    values.map { |number| number * 2 }
  end
  worker.send(numbers)
  worker.take
end
```

Ο worker παίρνει δεδομένα από receive και δεν κλείνει πάνω από το numbers της εξωτερικής μεθόδου. Η παραλλαγή μετακινεί μόνο δικό της δοκιμαστικό String, ώστε η απώλεια πρόσβασης να είναι ρητό μέρος της άσκησης.

Η παραλλαγή

Σε δικό σου προσωρινό String σύγκρινε shareable? πριν/μετά το freeze. Έπειτα μετακίνησε άλλο String με move: true και επιβεβαίωσε ότι πρόσβαση στον αποστολέα προκαλεί Ractor::MovedError.

```ruby
text = "fixed"
before = Ractor.shareable?(text)
text.freeze
after = Ractor.shareable?(text)
worker = Ractor.new { Ractor.receive.upcase }
payload = "hello"
worker.send(payload, move: true)
result = worker.take
moved = begin
  payload.length
  false
rescue Ractor::MovedError
  true
end
[before, after, result, moved]
```

Αποτέλεσμα: `[false, true, "HELLO", true]`.