Μάντεψε τον Αριθμό

3

29 Αυγούστου 2016
11:37

Ένα από τα πρώτα προγράμματα που συνηθίζεται να φτιάχνουν οι μαθητευόμενοι προγραμματιστές είναι ένα παιχνίδι στο οποίο ο παίκτης προσπαθεί να μαντέψει τον *μυστικό αριθμό* που έχει "σκεφτεί" ο υπολογιστής ή το αντίστροφο. Υπάρχουν πολλοί καλοί λόγοι που αυτό το παιχνίδι είναι μια τόσο δημοφιλής επιλογή για τους αρχάριους: το πρόγραμμα που προκύπτει δεν είναι ιδιαίτερα περίπλοκο, αν και συνδυάζει όλες τις βασικές αλγοριθμικές έννοιες, ενώ το παιχνίδι καθεαυτό είναι πολύ διασκεδαστικό.

Έννοιες: δομή επιλογής, δομή επανάληψης, αναζήτηση.

Έχω Ένα Μυστικό

Ας ξεκινήσουμε ονομάζοντας τον μυστικό αριθμό `secret` και δίνοντάς του μια τυχαία τιμή από το `1` μέχρι και το `32`.

```
import random
# δημιουργία τυχαίου μυστικού αριθμού
secret = random.randint(1,32)
```

Εισάγουμε τη βιβλιοθήκη `random` για να χρησιμοποιήσουμε την συνάρτηση `randint()`, που παράγει τυχαίους ακέραιους εντός καθορισμένων ορίων.

Τώρα θα ζητήσουμε από το χρήστη να μαντέψει τον μυστικό αριθμό.

```
# εμφάνιση προτροπής και ανάγνωση αριθμού
print("Μάντεψε τον αριθμό: 1 - 32")
number = int(input())
```

Κάθε τιμή έχει συγκεκριμένο *τύπο*. Η τιμή που επιστρέφει η `input()` είναι *αλφαριθμητική*, είναι το κείμενο που πληκτρολόγησε ο χρήστης. Χρειάζεται να μετατρέψουμε την τιμή αυτή σε ακέραιο αριθμό, να της αλλάξουμε τον τύπο, και για την μετατροπή αυτή χρησιμοποιούμε την `int()`.

Στην Python, οι ακέραιες τιμές είναι τύπου `int`, ενώ οι αλφαριθμητικές είναι τύπου `str`. Κάθε αλφαριθμητική τιμή περικλείεται σε εισαγωγικά (μονά ή διπλά). Τα μηνύματα που εμφανίζουμε είναι αλφαριθμητικές τιμές, γι' αυτό περικλείονται σε εισαγωγικά.

Το Βρήκα;

Πώς μπορώ να ελέγξω αν ο χρήστης μάντεψε τον μυστικό αριθμό;

Με μια δομή επιλογής θα συγκρίνουμε τον αριθμό `number` που έδωσε ο χρήστης με τον μυστικό αριθμό `secret` και θα εκτελέσουμε διαφορετικές εντολές ανάλογα με το αποτέλεσμα της σύγκρισης.

```
# έλεγχος αριθμού και εμφάνιση μηνύματος
if number != secret:
    print("Λάθος.")
else:
    print("Σωστά!")
```

guess/src/guess.1.py

Με το `!=` ελέγχεται αν δύο τιμές είναι διαφορετικές.

Αν εκτελέσετε το πρόγραμμά σας και πληκτρολογήσετε έναν αριθμό τότε αυτό θα σας απαντήσει `"Σωστά!"` ή `"Λάθος."` Για να επαληθεύσετε ότι αυτή η απάντηση είναι ορθή, θα πρέπει να γνωρίζετε τον μυστικό αριθμό, πράγμα δύσκολο όταν αυτός καθορίζεται τυχαία.

Για να ελέγξετε λοιπόν ότι το πρόγραμμά σας λειτουργεί σωστά, θα μπορούσατε *προσωρινά* να εμφανίζετε τον μυστικό αριθμό.

```
# ΓΙΑ ΕΛΕΓΧΟ: εμφάνιση του μυστικού αριθμού
print(secret)
```

Εναλλακτικά, θα μπορούσατε να ορίσετε έναν συγκεκριμένο μυστικό αριθμό της επιλογής σας, αντί να παράγετε έναν τυχαίο:

```
# ΓΙΑ ΕΛΕΓΧΟ: ορισμός μυστικού αριθμού
secret = 13
```

Έτσι θα μπορέσετε να επιβεβαιώσετε αν το πρόγραμμά σας λειτουργεί σωστά όταν του δίνετε έναν αριθμό. Στην τελική εκδοχή του προγράμματος, όταν οι έλεγχοι έχουν ολοκληρωθεί, θα πρέπει να αφαιρέσετε αυτές τις εντολές.

Γύρω-Γύρω Όλοι

Πως γίνεται η διαδικασία να επαναλαμβάνεται, έτσι ώστε ο χρήστης να έχει περισσότερες προσπάθειες για να μαντέψει τον αριθμό;

Οι εντολές που διαβάζουν έναν αριθμό από το χρήστη και τον συγκρίνουν με τον μυστικό αριθμό θα πρέπει να τοποθετηθούν μέσα σε μια *επαναληπτική δομή*.

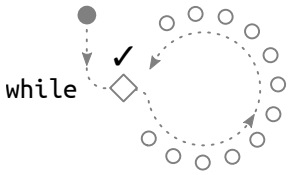
```
# επανάληψη:
#   δεν τερματίζεται, συνθήκη πάντα αληθής
while True:
    # εμφάνιση προτροπής και ανάγνωση αριθμού
    print("Μάντεψε τον αριθμό: 1 - 32")
    number = int(input())
    # έλεγχος αριθμού και εμφάνιση μηνύματος
    if number != secret:
        print("Λάθος.")
    else:
        print("Σωστά!")
```

guess/src/guess.2.py

Η επαναληπτική δομή **`while`** συνοδεύεται από μια *συνθήκη συνέχειας*. Η συνθήκη ελέγχεται στην αρχή κάθε κύκλου της επανάληψης και μπορεί να είναι αληθής (**`True`**) ή ψευδής (**`False`**). Όσο η συνθήκη είναι αληθής, η επανάληψη συνεχίζεται για άλλον έναν κύκλο.

Μην παραλείπετε το σύμβολο `:` μετά την συνθήκη.

Οι εντολές που ακολουθούν τη **`while`** είναι στοιχισμένες δεξιότερα. Η στοίχιση αυτή υποδηλώνει ότι αυτές οι εντολές θα επαναλαμβάνονται. Η πρώτη εντολή μετά τη **`while`** που δεν θα είναι στοιχισμένη δεξιότερα δεν θα επαναλαμβάνεται, αλλά θα εκτελεστεί μόνο μια φορά, όταν η επανάληψη τερματιστεί.



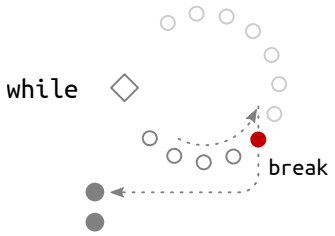
Σχήμα 3.1: Η τετριμμένη συνθήκη **`True`** που ελέγχεται από τη **`while`** είναι πάντα αληθής, κι έτσι η συγκεκριμένη επανάληψη δεν πρόκειται να διακοπεί: οι εντολές της **`while`** εκτελούνται επ' άπειρον.

Σταματήστε Να Κατέβω

Δεν θα πρέπει να σταματά η επανάληψη όταν βρεθεί ο αριθμός;

Οι εντολές που τοποθετήθηκαν στην επαναληπτική δομή εκτελούνται *συνεχώς* και δεν διακόπτονται ούτε καν όταν ο χρήστης μαντέψει τον μυστικό αριθμό. Ένας απλός τρόπος να τερματιστεί η επανάληψη είναι με την προσθήκη της εντολής **`break`** όταν ο παίκτης εντοπίσει τον μυστικό αριθμό.

```
# επανάληψη:
#   τερματίζεται (με break) όταν βρεθεί ο αριθμός
while True:
    # εμφάνιση προτροπής και ανάγνωση αριθμού
    print("Μάντεψε τον αριθμό: 1 - 32")
    number = int(input())
    # έλεγχος αριθμού και εμφάνιση μηνύματος
    if number != secret:
        print("Λάθος.")
    else:
        print("Σωστά!")
        # άμεση έξοδος από την επανάληψη
        break
```



Σχήμα 3.2: Η **break** διακόπτει και τερματίζει *άμεσα* τον κύκλο της επανάληψης, χωρίς να ελεγχθεί η συνθήκη συνέχειας. Οι εντολές μετά την **break** αγνοούνται και η εκτέλεση συνεχίζεται από την πρώτη εντολή που ακολουθεί την επανάληψη.

Για να κατανοήσετε καλύτερα τι σημαίνει η άμεση διακοπή της επανάληψης όταν εκτελεστεί η **break**, δοκιμάστε να τοποθετήσετε την εντολή `print("Σωστά!")` *μετά* την **break**. Θα διαπιστώσετε ότι το μήνυμα δεν θα εμφανιστεί ποτέ.

Εφόσον το παιχνίδι θα πρέπει να συνεχίζεται όσο ο χρήστης δεν έχει μαντέψει τον μυστικό αριθμό, ίσως αναρωτιέστε γιατί η συνθήκη συνέχειας δεν έχει απλά την εξής μορφή:

```
while number != secret:
```

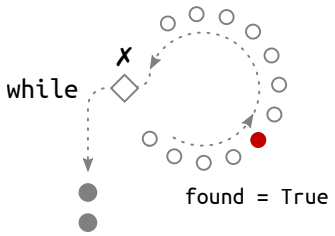
Αυτή είναι πράγματι μια εξαιρετική παρατήρηση, και θα έχετε την ευκαιρία να τη διερευνήσετε στην άσκηση 3.2. Προς το παρόν, θα συνεχίσουμε τροποποιώντας την τρέχουσα εκδοχή με τέτοιο τρόπο ώστε η διακοπή της επανάληψης να μην βασίζεται στην **break**, αλλά σε μια *λογική* μεταβλητή που θα ελέγχεται στη συνθήκη συνέχειας της επανάληψης.

```
# ο μυστικός αριθμός δεν έχει εντοπιστεί
found = False
# επανάληψη:
#   τερματίζεται όταν βρεθεί ο αριθμός
while not found:
    # εμφάνιση προτροπής και ανάγνωση αριθμού
    print("Μάντεψε τον αριθμό: 1 - 32")
    number = int(input())
    # έλεγχος αριθμού και εμφάνιση μηνύματος
    if number != secret:
        print("Λάθος.")
    else:
        print("Σωστά!")
        # ο μυστικός αριθμός εντοπίστηκε
        found = True
```

guess/src/guess.3.py

Υπάρχουν δύο *λογικές* τιμές: **True** και **False**, οι οποίες είναι τύπου `bool`.

Το **not** χρησιμοποιείται πριν από μια συνθήκη και *αντιστρέφει* την τιμή της: όταν μια συνθήκη είναι ψευδής τότε η αντίστροφή της είναι αληθής και το ανάποδο.



Σχήμα 3.3: Όταν η `found` πάρει τιμή **True**, η επανάληψη *δεν* θα διακοπεί άμεσα, γιατί η συνθήκη **not** `found` της **while** δεν ελέγχεται συνεχώς. Η επανάληψη θα διακοπεί όταν ελεγχθεί η συνθήκη της **while**, δηλαδή αφού ολοκληρωθεί ο κύκλος της επανάληψης.

Η μεταβλητή `found` λειτουργεί ως *σημαία* και χρησιμοποιείται για να "θυμάται" το πρόγραμμά μας αν ο παίκτης έχει βρει τον μυστικό αριθμό. Αρχικά ορίζεται ως ψευδής και όσο εξακολουθεί να είναι ψευδής η επανάληψη συνεχίζεται. Όταν ο παίκτης βρει τον αριθμό, η τιμή της `found` αλλάζει σε αληθής και η επανάληψη τερματίζεται· όχι άμεσα, όπως με την **break**, αλλά όταν ελεγχθεί η `found` στη

συνθήκη της **while**, δηλαδή μετά το τέλος της τρέχοντος κύκλου της επανάληψης.

Το Μέτρημα

Το παιχνίδι δεν έχει ιδιαίτερο ενδιαφέρον αν ο χρήστης διαθέτει απεριόριστες ευκαιρίες να μαντέψει τον αριθμό. Πως γίνεται να του δώσουμε μόνο ένα μικρό πλήθος προσπαθειών;

Σε κάθε κύκλο του παιχνιδιού, θα πρέπει το πρόγραμμά μας να "γνωρίζει" πόσες προσπάθειες απομένουν στο χρήστη. Αυτό είναι απαραίτητο γιατί ο χρήστης θα πρέπει να ενημερώνεται σχετικά, αλλά κυρίως επειδή το ίδιο το πρόγραμμα χρειάζεται αυτή την πληροφορία για να γνωρίζει πότε θα τερματιστεί το παιχνίδι. Θα καταμετρούμε τις προσπάθειες που απομένουν με τη μεταβλητή `tries`.

Αρχικά, ας δώσουμε στο χρήστη ένα αυθαίρετο πλήθος προσπαθειών, π.χ. τέσσερις, προσθέτοντας την ακόλουθη εντολή πριν την έναρξη της επανάληψης.

```
6  # ορισμός μέγιστου πλήθους προσπαθειών
7  tries = 4
```

Η απόδοση αρχικής τιμής σε μια μεταβλητή, όπως έγινε εδώ για την μεταβλητή `tries`, ονομάζεται *αρχικοποίηση*. Στη συνέχεια του παιχνιδιού, καθώς ο παίκτης προσπαθεί να μαντέψει τον αριθμό, η τιμή της `tries` θα μειώνεται κατά μια μονάδα σε κάθε γύρο.

Ο τερματισμός του παιχνιδιού εξαρτάται τώρα και από το πλήθος των προσπαθειών που απομένουν. Στην αρχή κάθε κύκλου της επανάληψης θα ελέγχεται αν ο μυστικός αριθμός έχει βρεθεί, αλλά και αν η μεταβλητή `tries` είναι θετική, δηλαδή αν απομένουν ακόμα προσπάθειες. Αν κάποια από αυτές τις συνθήκες δεν ισχύει, η επαναληπτική διαδικασία διακόπτεται.

```
8  # επανάληψη: τερματίζεται όταν
9  #   βρεθεί ο αριθμός ή εξαντληθούν οι προσπάθειες
10 while not found and tries > 0:
```

Το **and** χρησιμοποιείται για τη σύζευξη δύο συνθηκών. Η σύζευξη είναι αληθής μόνο όταν *και οι δύο* συζευγμένες συνθήκες είναι αληθείς.

Σε κάθε γύρο του παιχνιδιού, ο χρήστης ενημερώνεται με ένα μήνυμα για το πλήθος των προσπαθειών που απομένουν, ενώ στη συνέχεια η ποσότητα αυτή (η μεταβλητή `tries`) μειώνεται κατά μία μονάδα.

```
11     # εμφάνιση και μείωση προσπαθειών
12     print("Απομένουν", tries, "προσπάθειες.")
13     tries = tries - 1
```

Η τελευταία εντολή μπερδεύει πολλούς αρχάριους γιατί την ερμηνεύουν σαν μαθηματική ισότητα, ενώ δεν είναι. Για να γίνει κατανοητή, πρέπει να διαβαστεί ως εξής: υπολόγισε την τιμή της παράστασης `tries - 1` κι ονόμασε το αποτέλεσμα `tries`. Η νέα τιμή της μεταβλητής `tries` υπολογίζεται με βάση την τρέχουσα τιμή της, την οποία και αντικαθιστά.

Προσέξτε πως όταν η `tries` γίνει μηδέν, η επανάληψη *δεν* θα τερματιστεί άμεσα, αφού η συνθήκη συνέχειας δεν ελέγχεται συνεχώς, αλλά μόνο στην αρχή κάθε νέου κύκλου. Επομένως, ο κύκλος των εντολών θα ολοκληρωθεί και μόνο τότε θα διαπιστωθεί ότι η `tries` μηδενίστηκε, με αποτέλεσμα τη διακοπή της επανάληψης.

Όταν η επανάληψη διακοπεί, το παιχνίδι θα έχει τελειώσει. Δεν θα γνωρίζουμε όμως *για ποιο λόγο* τελείωσε. Μάντεψε ο παίκτης τον μυστικό αριθμό ή απλά εξαντλήθηκαν οι προσπάθειές του; Στη δεύτερη περίπτωση ο παίκτης θα έχει χάσει και είναι απαραίτητο να εμφανίζουμε ένα μήνυμα που θα τον ενημερώνει ποιος ήταν ο μυστικός αριθμός (για να μην πιστεύει ότι τον κλέβουμε).

```
24 # μετά την επανάληψη
25 # εμφάνιση μηνύματος αν δεν έχει βρεθεί ο αριθμός.
26 if not found:
27     print("Ήταν ο", secret)
```

guess/src/guess.4.py

Οι εντολές αυτές είναι στοιχισμένες πιο αριστερά από τις προηγούμενες. Αυτό υποδηλώνει ότι δεν ανήκουν στην επανάληψη, αντίθετα είναι οι πρώτες εντολές που θα εκτελεστούν όταν η επανάληψη διακοπεί.

Περισσότερη Πληροφορία

Θα ήθελα η απάντηση που δίνεται στο χρήστη να τον βοηθάει λίγο περισσότερο να βρει τον αριθμό. Αντί να ενημερώνεται αν τον μάντεψε ή όχι, θα μπορούσε να κατευθύνεται να ψάξει προς τα πάνω ή προς τα κάτω.

Μέχρι στιγμής, σε κάθε προσπάθεια το πρόγραμμα ελέγχει αν ο αριθμός που έδωσε ο χρήστης ταυτίζεται με τον μυστικό αριθμό. Τώρα θα επεκτείνουμε αυτόν τον έλεγχο, έτσι ώστε να ελέγχεται αν ο αριθμός του χρήστη είναι μεγαλύτερος ή μικρότερος από τον μυστικό αριθμό. Σε κάθε περίπτωση θα εμφανίζεται διαφορετικό μήνυμα, έτσι ώστε ο χρήστης να *κατευθύνεται* προς τον μυστικό αριθμό.

```
17     # έλεγχος αριθμού και εμφάνιση μηνύματος
18     if number > secret:
19         print("Λάθος. Είναι μικρότερος.")
20     elif number < secret:
21         print("Λάθος. Είναι μεγαλύτερος.")
22     else:
23         print("Σωστά!")
24         # ο μυστικός αριθμός εντοπίστηκε
25         found = True
```

guess/src/guess.5.py

Το **`elif`** σημαίνει **`else if`**. Χρησιμοποιείται στη δομή *πολλαπλής επιλογής*, δηλαδή όταν χρειάζεται να διακρίνουμε ανάμεσα σε περισσότερες από δύο περιπτώσεις.

Κάθε **`elif`**, όπως και η **`if`**, συνοδεύεται από μια *συνθήκη*. Μετά από μια **`if`** μπορούμε να χρησιμοποιήσουμε όσες διαδοχικές **`elif`** είναι απαραίτητες. Οι συνθήκες ελέγχονται διαδοχικά, η μία μετά την άλλη, μέχρι να βρεθεί μία που να είναι αληθής, οπότε και εκτελούνται οι αντίστοιχες εντολές, ενώ οι συνθήκες που την ακολουθούν παρακάμπτονται.

Σκεφτείτε πόση περισσότερη *πληροφορία* παρέχεται στο χρήστη με αυτή την μικρή αλλαγή. Προηγουμένως, κάθε αποτυχημένη προσπάθεια απέκλειε έναν από τους υποψήφιους αριθμούς. Τώρα μια αποτυχημένη προσπάθεια αποκλείει και *όλους* τους αριθμούς που είναι μικρότεροι ή μεγαλύτεροι από αυτόν που επέλεξε ο χρήστης.

Ακόμα Περισσότερη Πληροφορία

Μετά από μερικές προσπάθειες γίνεται δύσκολο για το χρήστη να θυμάται που ακριβώς έχει περιορίσει τον μυστικό αριθμό, δηλαδή από ποιον αριθμό είναι μεγαλύτερος και από ποιον μικρότερος. Πως γίνεται αυτή την πληροφορία να τη "θυμάται" το πρόγραμμα (αντί για τον χρήστη) και να τον ενημερώνει κατάλληλα για να τον βοηθά στις επιλογές του;

Ο,τιδήποτε είναι ανάγκη να "θυμάται" το πρόγραμμα αποθηκεύεται σε μεταβλητές. Θα χρησιμοποιήσουμε λοιπόν την μεταβλητή `low` για την ελάχιστη τιμή που θα μπορούσε να λάβει ο μυστικός αριθμός και την μεταβλητή `high` για την αντίστοιχη μέγιστη τιμή.

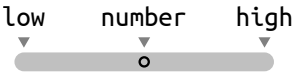
Σχήμα 3.4: Ο μυστικός αριθμός βρίσκεται κάπου ανάμεσα στα όρια `low` και `high`.

Στην αρχή του προγράμματος θα πρέπει να αποδώσουμε στις μεταβλητές αυτές μια αρχική τιμή – θα τις *αρχικοποιήσουμε*.

```
2  # οι μεταβλητές low και high είναι τα όρια
3  # ανάμεσα στα οποία βρίσκεται ο μυστικός αριθμός
4  low = 1
5  high = 32
6  # δημιουργία τυχαίου μυστικού αριθμού
7  secret = random.randint(low,high)
```

Μέσα στην επανάληψη εμφανίζουμε τις τιμές αυτών των ορίων, για να βοηθήσουμε το χρήστη στην επιλογή αριθμού.

```
18     # εμφάνιση προτροπής και ανάγνωση αριθμού
19     print("Μάντεψε τον αριθμό:", low , "-", high)
20     number = int(input())
```



Σχήμα 3.5: Επιλέγεται ένας αριθμός `number` μεταξύ των `low` και `high`.

Θα φροντίσουμε όμως οι μεταβλητές `low` και `high` να τροποποιούνται κατάλληλα μετά από κάθε λανθασμένη επιλογή του χρήστη.

Συγκεκριμένα, όταν ο χρήστης επιλέξει έναν αριθμό `number` που είναι μεγαλύτερος από τον μυστικό, τότε η μεταβλητή `high` πρέπει να τροποποιηθεί κατάλληλα: ο μυστικός αριθμός δεν μπορεί να είναι μεγαλύτερος από το `number-1`.

Σχήμα 3.6: Αν ο μυστικός αριθμός είναι μικρότερος από το `number` τότε και το άνω όριο `high` πρέπει να γίνει μικρότερο από το `number`.

```
21     # έλεγχος αριθμού και εμφάνιση μηνύματος
22     if number > secret:
23         print("Λάθος. Είναι μικρότερος.")
24         high = number - 1
```

Όταν ο χρήστης επιλέξει έναν αριθμό `number` που είναι μικρότερος από τον μυστικό, τότε η μεταβλητή `low` πρέπει να τροποποιηθεί: ο μυστικός αριθμός δεν μπορεί να είναι μικρότερος από το `number+1`.

Σχήμα 3.7: Αν ο μυστικός αριθμός είναι μεγαλύτερος από το `number` τότε και το κάτω όριο `low` πρέπει να γίνει μεγαλύτερο από το `number`.

```
25     elif number < secret:
26         print("Λάθος. Είναι μεγαλύτερος.")
27         low = number + 1
```

guess/src/guess.6.py

Εξαρτήματα κι Αυτοματισμοί

Το καημένο το πρόγραμμά μου κάνει όλη τη βαρετή δουλειά. Δε γίνεται να το αφήσω να παίξει κι αυτό λιγάκι;

Το σημείο στο οποίο ο παίκτης αλληλεπιδρά με το παιχνίδι είναι το σημείο στο οποίο του ζητείται να μαντέψει τον μυστικό αριθμό. Ο παίκτης ενημερώνεται για τα όρια ανάμεσα στα οποία βρίσκεται ο μυστικός αριθμός και, με βάση αυτές τις τιμές, επιλέγει έναν αριθμό.

```
    # εμφάνιση προτροπής και ανάγνωση αριθμού
    print("Μάντεψε τον αριθμό:", low , "-", high)
    number = int(input())
```

Ας κατασκευάσουμε λοιπόν ένα υποπρόγραμμα το οποίο θα κάνει αυτή ακριβώς τη δουλειά: θα δέχεται σαν *παραμέτρους* τα όρια ανάμεσα στα οποία βρίσκεται ο μυστικός αριθμός και θα ζητάει από το χρήστη να επιλέξει έναν αριθμό μέσα σε αυτά τα όρια.

```
2  def readNumber(a,b):
3      """ ζητάει από το χρήστη έναν αριθμό
4      μεταξύ των a και b και τον επιστρέφει.
5      a, b: όρια για τον αριθμό (δεν ελέγχονται)
6      """
7      # εμφάνιση προτροπής και ανάγνωση αριθμού
8      print("Μάντεψε τον αριθμό:", a , "-", b)
9      num = int(input())
10     # επιστροφή αριθμού
11     return num
```

Οι μεταβλητές `a`, `b` και `num` είναι τοπικές και υφίστανται μόνο καθόσο εκτελείται το υποπρόγραμμα.

Στο σημείο του κύριου προγράμματος όπου ζητείται η είσοδος αριθμού από το χρήστη μπορούμε τώρα να καλέσουμε το υποπρόγραμμα, παρέχοντας τις κατάλληλες παραμέτρους.

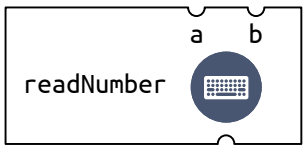
```
28     # επιλογή αριθμού από το χρήστη
29     number = readNumber(low,high)
```

guess/src/guess.7.py

Κατά την κλήση αυτή, οι μεταβλητές του κύριου προγράμματος `low` και `high` περνούν τις τιμές τους στις αντίστοιχες τοπικές μεταβλητές `a` και `b`, όπως φαίνεται και στο σχήμα 3.9.

Ας αντικαταστήσουμε τώρα την συνάρτηση `readNumber()` με μια άλλη που θα επιλέγει μόνη της έναν αριθμό, παίζοντας το ρόλο του χρήστη. Όμως, ποιον αριθμό θα πρέπει να επιλέγει; Εσείς, όταν δοκιμάζετε το πρόγραμμά σας, αναλαμβάνοντας το ρόλο του χρήστη, επιλέγετε τυχαία αριθμούς ή μήπως χρησιμοποιείτε συγκεκριμένη τακτική; Μπορείτε να περιγράψετε με σαφήνεια πως σκέφτεστε για να επιλέξετε τον επόμενο αριθμό που θα δοκιμάσετε;

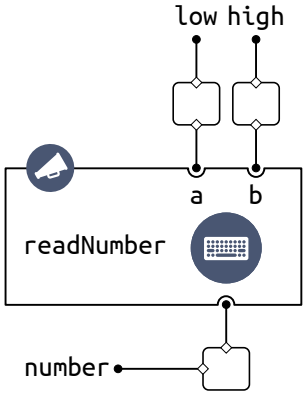
Το υποπρόγραμμα που ακολουθεί δέχεται σαν παραμέτρους τα όρια



Σχήμα 3.8: Αναπαράσταση της συνάρτησης `readNumber()`, η οποία δέχεται σαν παραμέτρους δύο ακέραιες τιμές `a` και `b`, ζητά από το χρήστη να πληκτρολογήσει μια τιμή που βρίσκεται ανάμεσά τους και την επιστρέφει.

Ο ορισμός μιας συνάρτησης ξεκινά με τη δεσμευμένη λέξη **`def`** κι ακολουθείται από το όνομα της συνάρτησης και τις παραμέτρους της, σε παρενθέσεις.

Οι εντολές που ακολουθούν την πρώτη γραμμή είναι στοιχισμένες δεξιότερα. Η στοίχιση αυτή υποδηλώνει ότι οι εντολές αυτές θα εκτελεστούν όταν *κληθεί* η συνάρτηση.



Σχήμα 3.9: Η κλήση της συνάρτησης `readNumber()` από το κύριο πρόγραμμα. Οι τιμές των `low` και `high` χρησιμοποιούνται ως τιμές των παραμέτρων a και b. Η τιμή που επιστρέφεται αποθηκεύεται στη `number`.

ανάμεσα στα οποία βρίσκεται ο μυστικός αριθμός κι επιλέγει τον αριθμό που βρίσκεται *στο μέσο του διαστήματος* ανάμεσα στα όρια.

```
2  def midNumber(a,b):
3      """ επιλέγει τον μεσαίο αριθμό
4      μεταξύ των a και b και τον επιστρέφει.
5      a, b: όρια για τον αριθμό
6      """
7      # εμφάνιση προτροπής
8      print("Μάντεψε τον αριθμό:", a , "-", b)
9      # υπολογισμός μεσαίου αριθμού
10     num = (a + b) // 2
11     print("Ο υπολογιστής επιλέγει:", num)
12     # επιστροφή αριθμού
13     return num
```

Σχήμα 3.10: Αναπαράσταση της συνάρτησης `midNumber()`, η οποία δέχεται σαν παραμέτρους δύο ακέραιες τιμές `a` και `b` και επιστρέφει την τιμή που βρίσκεται στο μέσο τους.

Το σύμβολο `//` χρησιμοποιείται για να υπολογιστεί το πηλίκο της *ακέραιας διαίρεσης* (χωρίς δεκαδικά ψηφία). Χρειαζόμαστε αυτή την πράξη, αντί για την συνηθισμένη διαίρεση, επειδή θέλουμε ο αριθμός `number` να είναι πάντα ακέραιος.

Τώρα η κλήση προς τη `readNumber`, η οποία ζητούσε αριθμό από το χρήστη, αντικαθίσταται με μια κλήση προς την `midNumber`, η οποία επιλέγει η ίδια τον αριθμό που θα ελεγχθεί. Παρατηρήστε την ευκολία με την οποία αντικαθιστουμε το ένα "εξάρτημα" μ' ένα άλλο, εφόσον και τα δύο χρησιμοποιούνται για την επιλογή του επόμενου αριθμού που θα δοκιμαστεί.

```
30     # επιλογή αριθμού από το ίδιο το πρόγραμμα
31     number = midNumber(low,high)
```

guess/src/guess.8.py

Στη φάση αυτή δεν υπάρχει πλέον είσοδος από τον χρήστη. Ο μυστικός αριθμός καθορίζεται με τυχαίο τρόπο και στη συνέχεια το ίδιο το πρόγραμμα (αναλαμβάνοντας και το ρόλο του δεύτερου παίκτη) προσπαθεί *με συστηματικό τρόπο* να τον μαντέψει.

Πλήρες Τελικό Πρόγραμμα

```
1  import random

2  def readNumber(a,b):
3      """ ζητάει από το χρήστη έναν αριθμό
4      μεταξύ των a και b και τον επιστρέφει.
5      a, b: όρια για τον αριθμό (δεν ελέγχονται)
6      """
7      # εμφάνιση προτροπής και ανάγνωση αριθμού
8      print("Μάντεψε τον αριθμό:", a , "-", b)
9      num = int(input())
10     # επιστροφή αριθμού
11     return num
```

...συνεχίζεται στην επόμενη σελίδα.

```
12  # οι μεταβλητές low και high είναι τα όρια
13  # ανάμεσα στα οποία βρίσκεται ο μυστικός αριθμός
14  low = 1
15  high = 32
16  # δημιουργία τυχαίου μυστικού αριθμού
17  secret = random.randint(low,high)
18  # ο μυστικός αριθμός δεν έχει εντοπιστεί
19  found = False
20  # ορισμός μέγιστου πλήθους προσπαθειών
21  tries = 4
22  # επανάληψη: τερματίζεται όταν
23  #   βρεθεί ο αριθμός ή εξαντληθούν οι προσπάθειες
24  while not found and tries > 0:
25      # εμφάνιση και μείωση προσπαθειών
26      print("Απομένουν", tries, "προσπάθειες.")
27      tries = tries - 1
28      # επιλογή αριθμού από το χρήστη
29      number = readNumber(low,high)
30      # έλεγχος αριθμού και εμφάνιση μηνύματος
31      if number > secret:
32          print("Λάθος. Είναι μικρότερος.")
33          high = number - 1
34      elif number < secret:
35          print("Λάθος. Είναι μεγαλύτερος.")
36          low = number + 1
37      else:
38          print("Σωστά!")
39          # ο μυστικός αριθμός εντοπίστηκε
40          found = True
41  # μετά την επανάληψη
42  # εμφάνιση μηνύματος αν δεν έχει βρεθεί ο αριθμός.
43  if not found:
44      print("Ήταν ο",secret)
```

guess/src/guess.final.py

Τροποποιήσεις – Επεκτάσεις

3.1 *Το τελικό πρόγραμμα ενημερώνει σε κάθε προσπάθεια για τα όρια ανάμεσα στα οποία βρίσκεται ο μυστικός αριθμός. Ωστόσο, ο χρήστης δεν είναι* υποχρεωμένος *να εισάγει έναν αριθμό που να βρίσκεται ανάμεσα σε αυτά τα όρια. Αν ο χρήστης δώσει έναν αριθμό εκτός ορίων τότε απλά θα χαραμίσει μια προσπάθεια. Παράλληλα όμως, θα φανεί κι ένα πρόβλημα: ένα από τα όρια θα τροποποιηθεί λανθασμένα. Επιβεβαιώστε το πρόβλημα και διορθώστε το, προσθέτοντας τις κατάλληλες εντολές.*

guess/exercises/guess-limits.py

3.2 *Διατυπώστε μια εναλλακτική εκδοχή του προγράμματος, στην οποία η* `while` *θα χρησιμοποιεί τη συνθήκη* `number != secret` *για να ελέγξει αν ο χρήστης δεν έχει βρει ακόμα τον μυστικό αριθμό και, επομένως, το παιχνίδι πρέπει να συνεχιστεί.*

Σημειώστε ότι την πρώτη φορά που η εκτέλεση του προγράμματος φτάσει στη **while**, την πρώτη φορά που θα ελεγχθεί η συνθήκη `number != secret`, η μεταβλητή `number` θα πρέπει να έχει ήδη μια τιμή.

Μια λύση είναι να της δώσουμε αρχικά, πριν την επανάληψη, μια *πλασματική* τιμή, για παράδειγμα:

```
# πλασματική αρχική τιμή για τον αριθμό του χρήστη
number = 0
# επανάληψη: τερματίζεται όταν
# βρεθεί ο αριθμός ή εξαντληθούν οι προσπάθειες
while number != secret and tries > 0:
```

Έτσι η `number` αποκτά τιμή και ταυτόχρονα είμαστε σίγουροι ότι η συνθήκη `number != secret` θα είναι αρχικά αληθής και η επανάληψη θα ξεκινήσει.

Όμως μια πολύ πιο ενδιαφέρουσα λύση (και αυτή είναι η ζητούμενη εδώ) είναι η αρχική τιμή για την `number` να προέρχεται από το χρήστη:

```
# επιλογή αριθμού από το χρήστη
number = readNumber(low,high)
# επανάληψη: τερματίζεται όταν
# βρεθεί ο αριθμός ή εξαντληθούν οι προσπάθειες
while number != secret and tries > 0:
```

Αυτή η προσέγγιση θα απαιτήσει σημαντική αναδιάταξη των εντολών για να λειτουργήσει το πρόγραμμα σωστά. Αξίζει τον κόπο γιατί θα σας οδηγήσει σε ένα μοτίβο που είναι πολύ κοινό σε προγράμματα με επανάληψη.

`guess/exercises/guess-nobreak.py`

3.3 *Στην τελευταία προσπάθεια, το πρόγραμμα εμφανίζει το άκομψο μήνυμα* `Απομένουν 1 προσπάθειες` *. Επίσης, αν η τελευταία προσπάθεια είναι αποτυχημένη, το πρόγραμμα ενημερώνει τον παίκτη αν ο μυστικός αριθμός είναι μικρότερος ή μεγαλύτερος, ενώ δεν υπάρχει πια λόγος για κάτι τέτοιο. Φαίνεται λοιπόν ότι η τελευταία προσπάθεια είναι ξεχωριστή και θα έπρεπε το πρόγραμμά μας να τη χειρίζεται με ιδιαίτερο τρόπο, κι όχι μαζί με τις υπόλοιπες.*

Τροποποιήστε το πρόγραμμα έτσι ώστε στην τελευταία προσπάθεια να εμφανίζεται το μήνυμα `Απομένει μία προσπάθεια` *και μετά την τελευταία προσπάθεια να εμφανίζεται είτε το* `Σωστά!` *, είτε ο μυστικός αριθμός, και τίποτε άλλο.*

Δεν είναι καλή ιδέα να χρησιμοποιήσετε μια **if**–**else** μέσα στην επανάληψη για να διαχωρίσετε την τελευταία προσπάθεια. Είναι προτιμότερο να τροποποιήστε τη συνθήκη της **while**, έτσι ώστε η επανάληψη να συνεχίζεται όταν `tries > 1`, αντί για `tries > 0`. Έτσι, όταν απομένει μια προσπάθεια η επανάληψη θα διακόπτεται. Προσθέστε *μετά* την επανάληψη τις κατάλληλες εντολές που αφορούν την τελευταία προσπάθεια.

`guess/exercises/guess-final.py`
`guess/exercises/guess-final-else.py`

3.4 *Να ορίσετε μια συνάρτηση* `computeNumber`*, η οποία επιστρέφει τον μεσαίο αριθμό μεταξύ των παραμέτρων* `a` *και* `b`*, όπως και η συνάρτηση* `midNumber`*, εκτός κι αν απομένει μόνο μια προσπάθεια. Στην περίπτωση αυτή, επιστρέφει έναν τυχαίο αριθμό μεταξύ των* `a` *και* `b`*.*

Η `midNumber` δέχεται ως παραμέτρους μόνο τα όρια `a` και `b` για να επιστρέψει τον αριθμό που βρίσκεται ανάμεσά τους. Η `computeNumber` χρειάζεται μια επιπλέον παράμετρο: το πλήθος των προσπαθειών που απομένουν. Επομένως, σε σχέση με την `midNumber`, θα πρέπει να τροποποιήσετε κατάλληλα τόσο τον ορισμό της συνάρτησης όσο και την κλήση της από το κύριο πρόγραμμα.

Η διαφορά ανάμεσα στην αρχική `midNumber` *και την* `computeNumber` *που θα υλοποιήσετε εδώ αφορά μόνο την τελευταία προσπάθεια και φαίνεται ασήμαντη. Όμως εχει ενδιαφέρον ν' αναρωτηθείτε τι πλεονέκτημα μπορεί να προσφέρει η χρήση της* `computeNumber`*. Τί κερδίζει ένας παίκτης που χρησιμοποιεί αυτή τη μέθοδο;*

`guess/exercises/guess-autorandom.py`

3.5 *Να τροποποιήσετε το πρόγραμμα έτσι ώστε εξωτερικά, από τη σκοπιά του χρήστη, να συμπεριφέρεται με τον ίδιο ακριβώς τρόπο αλλά στην πραγματικότητα να μην επιλέγει μυστικό αριθμό εκ των προτέρων. Το πρόγραμμα θα πρέπει και πάλι να διατηρεί τα όρια* `low` *και* `high` *ανάμεσα στα οποία ο χρήστης έχει περιορίσει τον μυστικό αριθμό. Όταν ο χρήστης δοκιμάζει έναν νέο αριθμό, το πρόγραμμα απαντά αν ο (ανύπαρκτος) μυστικός είναι μικρότερος ή μεγαλύτερος, προσπαθώντας να διατηρήσει το διάστημα ανάμεσα στα* `low` *και* `high` *όσο μεγαλύτερο γίνεται. Ο χρήστης ουσιαστικά κερδίζει μόνο αν τα* `low` *και* `high` *ταυτιστούν. Σε περίπτωση που εξαντληθούν οι προσπάθειες του παίκτη, το πρόγραμμα θα επιλέγει εκ των υστέρων έναν αριθμό ανάμεσα στα* `low` *και* `high` *και θ' ανακοινώνει πως αυτός ήταν ο μυστικός αριθμός.*

`guess/exercises/guess-cheat.py`

Ασκήσεις

3.6 *Να κατασκευάσετε ένα πρόγραμμα το οποίο θα ζητά από το χρήστη έναν αλφαριθμητικό κωδικό. Ο χρήστης θα έχει το πολύ τρεις προσπάθειες για να εισάγει τον κωδικό του. Αν εισάγει έναν σωστό κωδικό, το πρόγραμμα θα τον καλωσορίζει, ενώ αν εξαντλήσει ανεπιτυχώς τις προσπάθειές του, θα εμφανίζεται ένα σχετικό μήνυμα.*

`guess/exercises/password.py`

3.7 *Να κατασκευάσετε ένα πρόγραμμα στο οποίο* ο χρήστης *θα σκέφτεται έναν μυστικό αριθμό από το 1 μέχρι και το 32 και* το πρόγραμμα *θα προσπαθεί να τον μαντέψει μέσα σε 4 προσπάθειες το πολύ. Σε κάθε προσπάθεια, το πρόγραμμα θα επιλέγει έναν αριθμό και θα ρωτάει το χρήστη αν αυτός είναι ίσος, μικρότερος ή μεγαλύτερος από τον μυστικό αριθμό του.*

Πρόκειται για το ίδιο παιχνίδι με το οποίο ασχοληθήκαμε σ' αυτό το κεφάλαιο, αλλά με τους ρόλους παίκτη και προγράμματος αντεστραμμένους.

`guess/exercises/guess-user.py`

3.8 *Να κατασκευάσετε ένα πρόγραμμα το οποίο θα επιλέγει έναν μυστικό αριθμό από το 1 μέχρι και το 32 και ο χρήστης θα προσπαθεί να τον μαντέψει μέσα σε 4 προσπάθειες το πολύ. Σε κάθε προσπάθεια, ο χρήστης θα επιλέγει* δύο *αριθμούς που θ' αποτελούν την «παγίδα» του και το πρόγραμμα θα τον ενημερώνει αν ο μυστικός αριθμός βρίσκεται ανάμεσα στους αριθμούς της παγίδας, αν είναι μικρότερος ή μεγαλύτερος*

από αυτούς. Για να βρίσκεται ένας αριθμός μέσα στην παγίδα θα πρέπει να είναι τουλάχιστον ίσος με τον μικρότερο αριθμό της παγίδας και το πολύ ίσος με τον μεγαλύτερο. Όταν η παγίδα αποτελείται από δύο αριθμούς που είναι ίσοι μεταξύ τους και ταυτίζονται με τον μυστικό αριθμό τότε ο χρήστης έχει μαντέψει τον μυστικό αριθμό.

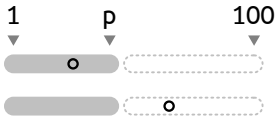
`guess/exercises/trap.py`

Όπως και στο «Μάντεψε τον Αριθμό», μπορείτε ν' αυτοματοποιήσετε την αναζήτηση του αριθμού, γράφοντας μια συνάρτηση αντίστοιχη της `midNumber()`*, η οποία επιλέγει σε κάθε γύρο την κατάλληλη παγίδα.*

`guess/exercises/trap-auto.py`

3.9 *Να γράψετε συνάρτηση με όνομα* `flipBiased`*, η οποία θα μιμείται τη ρίψη ενός* μεροληπτικού *νομίσματος και θα επιστρέφει είτε το* `0`*, είτε το* `1`*. Η συνάρτηση θα δέχεται μια παράμετρο* `p` *ή οποία θ' αντιστοιχεί στην πιθανότητα (επί τοις 100) να επιστραφεί το* `0`*.*

Ένας τρόπος υλοποίησης της λειτουργίας του μεροληπτικού νομίσματος είναι να επιλέγεται ένας τυχαίος αριθμός από το `1` μέχρι το `100`. Αν αυτός ο τυχαίος αριθμός δεν ξεπερνά το `p`, τότε επιστρέφεται το `0`, αλλιώς επιστρέφεται το `1`.

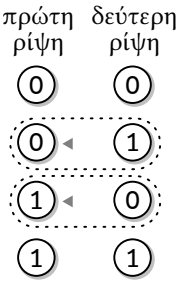


Αν ο τυχαίος αριθμός δεν ξεπερνά το `p`, επιστρέφεται το `0`. Σε διαφορετική περίπτωση, επιστρέφεται το `1`.

Ο Jon von Neumann σκέφτηκε έναν τρόπο να προσομοιώσει κανείς τη ρίψη ενός δίκαιου νομίσματος, ακόμα κι αν έχει στη διάθεσή του ένα μεροληπτικό νόμισμα. Η μέθοδος που πρότεινε είναι η εξής:

> *Ρίξε το μεροληπτικό νόμισμα δύο φορές. Αν το αποτέλεσμα σε κάθε ρίψη είναι διαφορετικό, τότε ανακοίνωσε το αποτέλεσμα της πρώτης ρίψης, αλλιώς επανάλαβε τη διαδικασία.*

Το αποτέλεσμα αυτής της μεθόδου ισοδυναμεί με το αποτέλεσμα της ρίψης ενός δίκαιου νομίσματος.



Όταν ρίχνουμε ένα νόμισμα δύο φορές υπάρχουν τέσσερα πιθανά ενδεχόμενα. Τα δύο από αυτά, όταν το αποτέλεσμα κάθε ρίψης είναι διαφορετικό, είναι *ισοπίθανα*, ακόμα κι όταν το νόμισμα είναι μεροληπτικό. Η μέθοδος του von Neumann επαναλαμβάνει τη ρίψη δύο νομισμάτων μέχρι να προκύψει ένα από τα δύο ισοπίθανα ενδεχόμενα, και επιστρέφει το αποτέλεσμα που σημειώνεται με το βελάκι, προσομοιώνοντας έτσι ένα δίκαιο νόμισμα.

Να γράψετε συνάρτηση με όνομα `flipFair`*, η οποία θα μιμείται τη ρίψη ενός δίκαιου νομίσματος και θα επιστρέφει είτε το* `0`*, είτε το* `1`*. Η συνάρτηση θα πρέπει να χρησιμοποιεί τη μέθοδο του von Neumann: θα υπολογίζει το αποτέλεσμά της χρησιμοποιώντας ένα μεροληπτικό νόμισμα, δηλαδή καλώντας την συνάρτηση* `flipBiased`*.*

Να φτιάξετε ένα πρόγραμμα που ελέγχει αν οι συναρτήσεις `flipBiased` *και* `flipFair` *λειτουργούν όπως θα περιμέναμε. Για κάθε συνάρτηση, το πρόγραμμα θα πρέπει να την καλεί επαναληπτικά και να υπολογίζει το ποσοστό των ρίψεων στις οποίες το αποτέλεσμα ήταν το* `0`*. Αν η* `flipBiased` *λειτουργεί σωστά, τότε το ποσοστό αυτό θα πρέπει να προσεγγίζει το* `p`*. Αν η* `flipFair` *λειτουργεί σωστά, τότε το ποσοστό αυτό θα πρέπει να προσεγγίζει το* `50`*.*

Καλό θα ήταν ο ίδιος ο χρήστης να παρέχει την τιμή της παραμέτρου `p` της `flipBiased` και το πλήθος των επαναλήψεων. Έτσι θα μπορέσετε κι εσείς, ως χρήστες, να πειραματιστείτε εύκολα με διάφορες τιμές. Όπως ισχύει σε κάθε προσομοίωση, το πλήθος των επαναλήψεων πρέπει να είναι μεγάλο (π.χ. τουλάχιστον 10000), για να είναι το αποτέλεσμα του ελέγχου αξιόπιστο.

`guess/exercises/flip-biased.py`

Αναζητηση *Όταν ψάχνουμε κάτι, όπως για παράδειγμα έναν μυστικό αριθμό, μιλάμε για* αναζήτηση. *Η αναζήτηση είναι κάτι που κάνουμε πολύ συχνά, είτε μόνοι μας, είτε με τη βοήθεια υπολογιστών. Ψάχνουμε για ένα όνομα στον τηλεφωνικό κατάλογο ή για την επόμενη κίνηση σ' ένα επιτραπέζιο παιχνίδι. Συχνά η αναζήτηση δεν είναι καθόλου απλή υπόθεση. Μπορεί ο χώρος στον οποίο ψάχνουμε να είναι αχανής. Μπορεί να μην ξέρουμε τι ακριβώς ψάχνουμε ή πως πρέπει να το περιγράψουμε. Για παράδειγμα, η μηχανή αναζήτησης της Google προσπαθεί μέσα σε μερικά δέκατα του δευτερολέπτου να συγκεντρώσει και να ταξινομήσει τις διευθύνσεις των ιστοσελίδων που πιθανώς μας ενδιαφέρουν, βασισμένη κάθε φορά σε μερικές λέξεις-κλειδιά που της παρέχουμε. Οι βιολόγοι αναζητούν ακολουθίες αμινοξέων μέσα στον γενετικό μας κώδικα, παρέχοντας μόνο ένα γενικό πρότυπο γιατί μπορεί να υπάρχουν αποδεκτές παραλλαγές και μεταλλάξεις. Με δεδομένο πάντως το πόσο σημαντικό πρόβλημα είναι η αναζήτηση, είναι ευτυχές ότι πρόκειται για ένα πρόβλημα που λύνεται συνήθως πολύ αποδοτικά.*

Δυαδικη Αναζητηση *Όταν αναζητούμε ένα αντικείμενο μέσα σε ένα σύνολο από* ταξινομημένα *στοιχεία τότε το καλύτερο που μπορούμε να κάνουμε είναι να ψάξουμε ακριβώς στη μέση. Αν το μεσαίο στοιχείο δεν είναι αυτό που ψάχνουμε τότε (επειδή τα στοιχεία είναι ταξινομημένα) γνωρίζουμε προς τα που πρέπει να συνεχίσουμε την αναζήτηση, αποκλείουμε μεμιάς τα άλλα μισά στοιχεία που βρίσκονται προς την αντίθετη κατεύθυνση και εφαρμόζουμε την ίδια διαδικασία στα στοιχεία που απομένουν. Με τον τρόπο αυτό, είτε θα εντοπίσουμε κάποια στιγμή το στοιχείο που αναζητούμε, είτε θα εξαντληθούν τα στοιχεία και θα φτάσουμε στο συμπέρασμα ότι αυτό που ψάχνουμε δεν υπάρχει ανάμεσά τους. Αυτή η μέθοδος ονομάζεται δυαδική αναζήτηση και είναι ουσιαστικά η διαδικασία που ακολουθήσαμε στο τελευταίο βήμα του παραδείγματος για να εντοπίσουμε τον μυστικό αριθμό.*

Διαιρει και Βασιλευε *Πρόκειται για μια στρατηγική επίλυσης προβλημάτων που στηρίζεται στον κατακερματισμό ενός προβλήματος σε μικρότερα υποπροβλήματα. Αυτά είναι συνήθως του ίδιου ή συναφούς τύπου με το αρχικό πρόβλημα και, στην καλύτερη περίπτωση, ίδιου μεγέθους μεταξύ τους. Τα υποπροβλήματα επιλύονται ξεχωριστά και, αν αυτό είναι απαραίτητο, οι λύσεις τους συνδυάζονται για να επιλυθεί το αρχικό πρόβλημα. Ουσιαστικά μια παρόμοια στρατηγική ακολουθούμε όταν το πρόβλημα που μας τίθεται είναι ο εντοπισμός του μυστικού αριθμού. Σε κάθε βήμα βρισκόμαστε αντιμέτωποι με το ίδιο πρόβλημα, αλλά τα όρια ανάμεσα στα οποία αναζητούμε τον αριθμό ολοένα και στενεύουν. Πολλοί σημαντικοί αλγόριθμοι στην Πληροφορική βασίζονται στη στρατηγική του διαίρει και βασίλευε.*
