

Ενότητα: Δυαδική Αναζήτηση

Σχέδιο Δραστηριότητας: Παιχνίδι: Βρες τον αριθμό

Εισαγωγή

Σκεφτείτε έναν αριθμό από το 1 έως το 1000 και απαντήστε στην ερώτηση:

Ο αριθμός που σκεφτήκατε είναι μεγαλύτερος ή μικρότερος από το 500;

Τι μπορείτε να εξάγετε από την απάντηση; Μεταξύ ποιων αριθμών πρέπει να ψάξετε τώρα για να βρείτε τον αριθμό;

Ποια θα είναι η επόμενη ερώτηση ώστε να περιορίσουμε ακόμα περισσότερο το χώρο αναζήτησης;

Η μέθοδος που σκιαγραφείται παραπάνω είναι γνωστή ως **Δυαδική αναζήτηση** και εκμεταλλεύεται τη διάταξη των στοιχείων του συνόλου, διαμερίζοντας κάθε φορά το σύνολο σε δυο ίσα μέρη, και εφαρμόζοντας πάλι την ίδια μέθοδο στο υποσύνολο στο οποίο ανήκει ο ζητούμενος αριθμός.

Δραστηριότητα : Μάντεψε τον αριθμό

Βήμα 1: Παραγωγή τυχαίων αριθμών

Πολλές φορές θέλουμε να παράγουμε αριθμούς με τυχαίο τρόπο. Η *βιβλιοθήκη random* περιέχει μια ποικιλία συναρτήσεων για αυτόν τον σκοπό. Δύο από αυτές που θα χρησιμοποιούμε κατά κόρον είναι η *randint* και η *randrange*, που επιστρέφουν τυχαίους ακέραιους αριθμούς εντός κάποιων ορίων. Τα όρια στην *randrange* ακολουθούν την ίδια λογική με την *range* της Python, ενώ η *randint* συμπεριλαμβάνει και το δεξιό άκρο.

```
import random
number = random.randint(1, 11)      # επιστρέφει έναν τυχαίο αριθμό στο [ 1, 10]
number = random.randrange(1,10)    # επιστρέφει έναν τυχαίο αριθμό στο [ 1, 9]
number = random.randrange(10)      # επιστρέφει έναν τυχαίο αριθμό στο [ 0, 9]
```

Άσκηση

Να γράψετε ένα πρόγραμμα σε Python το οποίο να παράγει τυχαία 6 ακέραιους αριθμούς στο διάστημα [1,45].

Βήμα 2: Ψάχνοντας στα τυφλά

Να συμπληρώσετε το παρακάτω πρόγραμμα έτσι ώστε να επιλέγει έναν αριθμό στην τύχη στο διάστημα [1,20] και στη συνέχεια επιτρέπει στον χρήστη να δοκιμάσει όσες προσπάθειες θέλει μέχρι να μαντέψει τον αριθμό. Ο χρήστης μαντεύει στην τύχη χωρίς κάποια συγκεκριμένη στρατηγική, με το πρόγραμμα να χρησιμοποιεί έναν μετρητή, τον *guesses*, για να μετράει τις προσπάθειες του χρήστη.

Στη συνέχεια να προσπαθήσετε να βρείτε τον αριθμό που “σκέφτηκε” ο υπολογιστής, με διαδοχικές μαντεψιές. Πόσες προσπάθειες χρειάζεστε στην χειρότερη περίπτωση;

```
import random
secret_number = _____
guesses = _____
found = _____
while not found :
    guess = input("Μάντεψε τον αριθμό : ")
    guesses = _____
    if _____ :
        print "Μπράβο το βρήκες με ", guesses, " προσπάθειες"
        found = _____
    else :
        print "Δυστυχώς δεν το βρήκες, Ξαναπροσπάθησε"
```

Στη συνέχεια να τροποποιήσετε το πρόγραμμα έτσι ώστε να επιτρέπει στον χρήστη μόνο 10 προσπάθειες.

Βήμα 3: Καταstrώνοντας τη στρατηγική

Θα τροποποιήσουμε το πρόγραμμα έτσι ώστε να δίνει μια βοήθεια στον παίκτη. Θα ελέγχει αν ο αριθμός που μάντεψε είναι μικρότερος ή μεγαλύτερος από τον μυστικό αριθμό και θα εμφανίζει κατάλληλο μήνυμα. Για να γίνει αυτό θα χρειαστεί να προσθέσουμε δυο ακόμα περιπτώσεις στη δομή επιλογής, όπως φαίνεται παρακάτω:

```
if guess == secret_number :
    print "Μπράβο το βρήκες με ", guesses, " προσπάθειες"
    found = True
else :
    if guess < secret_number:
        print "Είσαι χαμηλότερα"
    else :
        print " Είσαι υψηλότερα"
```

Έτσι αν ο αριθμός που ψάχνει ο παίκτης είναι μικρότερος από αυτόν που έδωσε, τότε ξέρει ότι πρέπει να ψάξει από εκεί και κάτω, με αποτέλεσμα το μέγεθος του προβλήματος να μειωθεί στο μισό.

Να τροποποιήσετε το πρόγραμμα που έχετε αναπτύξει έτσι ώστε να επιτρέπει στον παίκτη μέχρι 10 προσπάθειες. Αν τις ξεπεράσει να εμφανίζει μήνυμα ότι έχασε. Οι αριθμοί όμως τώρα θα είναι στο διάστημα [1,1000]. Τι πιστεύετε; Αρκούν 10 προσπάθειες για 1000 αριθμούς. Αν ναι τότε για 1.000.000 αριθμούς πόσες προσπάθειες πιστεύετε ότι χρειάζονται;


```
# Τώρα ο άνθρωπος σκέφτεται έναν αριθμό από 1 έως 1000
N = 1000
print "Σκέψου έναν αριθμό από το 1 έως το ", N
guesses = 0
found = False
first = _____
last = _____
while not found and guesses < 10 :
    mid = _____
    answer = raw_input("Είναι ο αριθμός ο " + str(mid) + " ? (N/O)" )
    guesses = guesses + 1
    if answer == "N" :
        found = _____
    else :
        answer = raw_input ("Είναι μικρότερος του " + str(mid) + " ? (N/O)" )
        if answer == "N" :
            _____ = _____
        else :
            _____ = _____
    if found == True :
        print "Κέρδισα!!! Το βρήκα με ", guesses, " προσπάθειες"
    else :
        print "Κέρδισες"
```

Στο παραπάνω πρόγραμμα στις μεταβλητές first και last καταχωρούμε τις θέσεις των δυο άκρων του διαστήματος και στην mid το μέσο του διαστήματος αναζήτησης.

Δυαδική αναζήτηση σε λίστα

Προηγουμένως χρησιμοποιήσαμε τον αλγόριθμο της δυαδικής αναζήτησης για να βρούμε έναν μυστικό αριθμό μέσα σε κάποια όρια. Ο αλγόριθμος βασίζεται στο γεγονός ότι οι αριθμοί είναι διατεταγμένοι κατά αύξουσα σειρά. Για να εφαρμοστεί ο ίδιος αλγόριθμος και σε άλλα δεδομένα, όπως για παράδειγμα σε ονόματα, θα πρέπει αυτά να είναι διατεταγμένα επίσης σε κάποιου είδους σειρά, όπως σε αλφαβητική. Για παράδειγμα με χρήση της δυαδικής αναζήτησης μπορούμε να βρούμε ένα όνομα στον τηλεφωνικό κατάλογο όπου όλα τα ονόματα είναι σε αλφαβητική σειρά. Έτσι μπορούμε να εφαρμόσουμε τη δυαδική αναζήτηση στα στοιχεία μιας λίστας τα οποία βρίσκονται σε κάποια λογική διάταξη είτε αυτά είναι αριθμοί είτε αλφαριθμητικά, όπως φαίνεται παρακάτω, όπου ψάχνουμε τον αριθμό 60 σε μια λίστα 16 αριθμών.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96

60 > 40

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96

60 < 63

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96

60 > 50

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96

Συμπληρώστε τα κενά ώστε η παρακάτω συνάρτηση να επιτελεί δυαδική αναζήτηση του στοιχείου key στα στοιχεία της λίστας array. Θεωρήστε ότι τα στοιχεία της λίστας είναι διατεταγμένα σε αύξουσα σειρά.

```
def binarySearch( array, key ) :
    first = 0
    last = _____
    found = False
    while _____ and not found :
        mid = ( first + last ) / 2
        if _____ :
            found = True
        elif _____ :
            first = mid + 1
        else :
            last = mid - 1
    return found
```

Στη συνέχεια να δώσετε στον διερμηνευτή της Python τις παρακάτω εντολές. Τι παρατηρείτε; Να εξηγήσετε τα αποτελέσματα.

```
>>> binarySearch( [1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ], 22)
>>> binarySearch( [1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ], 34)
>>> binarySearch( ["alpha", "beta", "gamma", "delta", "epsilon" ], "beta")
>>> binarySearch([False, False, False, False, True], False)
```