Τρίλιζα

5

Στο κεφάλαιο αυτό θα υλοποιήσουμε το γνωστό παιχνίδι της τρίλιζας. Θα αναπτύξουμε τον απαραίτητο κώδικα για ένα παιχνίδι δύο παικτών και μετά θα προσθέσουμε τις απαραίτητες πινελιές ώστε τον ρόλο του ενός παίκτη να τον αναλαμβάνει το ίδιο το πρόγραμμα. Προγραμματιστικά, θα εξασκηθούμε σε όλες τις έννοιες που έχουμε ήδη συναντήσει και θα δούμε πως μπορούμε να χρησιμοποιήσουμε *λίστες* και *πλειάδες* για την αναπαράσταση των δεδομένων μας. Η τρόπος που επιλέγουμε να αναπαραστήσουμε τα δεδομένα είναι αλληλένδετος με τον τρόπο που τα επεξεργαζόμαστε και επηρρεάζει άμεσα τη διαδικασία επίλυσης ενός προβλήματος.

23 Νοεμβρίου 2016
15:21

Έννοιες: υποπρογράμματα, αναπαραστάσεις, λίστες, πλειάδες

Το 1952, ο βρετανός Sandy Douglas έγραψε, στα πλαίσια του διδακτορικού του, ένα πρόγραμμα για τον υπολογιστή EDSAC που έπαιζε τρίλιζα. Αυτό το πρόγραμμα θεωρείται ένα από τα πρώτα βιντεοπαιχνίδια. Ο πίνακας της τρίλιζας προβαλλόνταν σε μια πρωτόγονη οθόνη και ο χρήστης επέλεγε την κίνησή του χρησιμοποιώντας το περιστρεφόμενο καντράν ενός τηλεφώνου.

Ένα τέτοιο πρόγραμμα για την τρίλιζα δεν είναι ιδιαίτερα δύσκολο να γραφτεί, η τρίλιζα είναι απλό παιχνίδι. Μάλιστα θα μπορούσε να χαρακτηριστεί ακόμα και βαρετό: όσο καλά και να παίξει κανείς, δεν μπορεί να κερδίσει τον αντίπαλό του παρά μόνο αν κάνει λάθη. Το «φυσιολογικό» αποτέλεσμα είναι η ισοπαλία. Κι όμως, οι άνθρωποι παίζουν τρίλιζα μ' ενθουσιασμό. Μάλιστα, όσο μεγαλύτερος είναι ο ενθουσιασμός, τόσο ευκολότερα χάνουν...

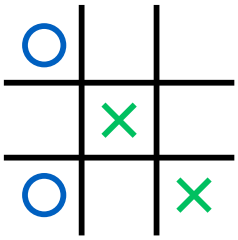
Αναπαραστάσεις

Πριν ξεκινήσουμε να γράψουμε έστω και μια γραμμή του προγράμματος, πρέπει να πάρουμε μια σημαντική απόφαση: ποια *αναπαράσταση* θα χρησιμοποιήσουμε για την τρίλιζα; Δηλαδή ποιος θα είναι ο τρόπος με τον οποίο το πρόγραμμά μας θα αποθηκεύει και θα διαχειρίζεται *εσωτερικά* τα περιεχόμενα των εννέα τετραγώνων του παιχνιδιού; Η απόφαση αυτή είναι κομβική και θα επηρρεάσει σημαντικά τη μορφή του προγράμματος που θα γράψουμε στη συνέχεια.

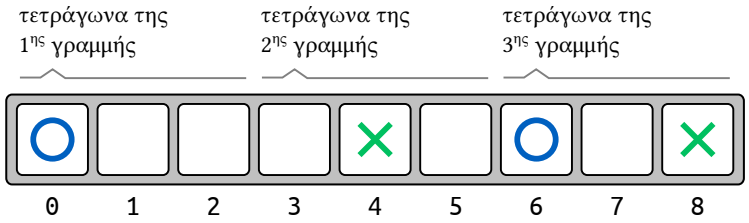
Δεν είναι βολικό ν' αποθηκεύσουμε το περιεχόμενο των εννέα τετραγώνων σε εννέα ξεχωριστές μεταβλητές. Θα καταλήξουμε με ένα δυσνόητο, εκτεταμένο πρόγραμμα με επαναλαμβανόμενα τμήματα. Αντιθέτως, θα θέλαμε τα εννέα τετράγωνα να αναπαρασταθούν ως μια *ενιαία, οργανωμένη συλλογή δεδομένων.*

Στο δικό μας πρόγραμμα θα χρησιμοποιήσουμε για τον σκοπό αυτό μια *λίστα*, δηλαδή μια συλλογή στοιχείων που είναι οργανωμένα *ακολουθιακά*, το ένα μετά το άλλο. Σε μια λίστα, τα στοιχεία είναι *αριθμημένα* για να μπορούμε μέσω αυτής της αρίθμησης να έχουμε πρόσβαση στο περιεχόμενό τους.

Ας εξετάσουμε ένα παράδειγμα, ένα συγκεκριμένο στιγμιότυπο του πίνακα της τρίλιζας:



Η σχετική λίστα θα αποτελείται από εννέα στοιχεία, σε κάθε ένα από τα οποία θα αποθηκεύεται το περιεχόμενο του αντίστοιχου τετραγώνου της τρίλιζας. Μια απεικόνιση της λίστας φαίνεται στο σχήμα 5.1 που ακολουθεί.



Σχήμα 5.1: Παράδειγμα αναπαράστασης ενός συγκεκριμένου στιγμιοτύπου του παιχνιδιού με τη χρήση *λίστας*. Σε κάθε στοιχείο της λίστας αποθηκεύεται το περιεχόμενο ενός τετραγώνου της τρίλιζας. Η συγκεκριμένη αντιστοιχία μεταξύ τετραγώνων και στοιχείων της λίστας είναι απλά μία από τις πολλές που θα μπορούσαν να χρησιμοποιηθούν.

Η αναπαράσταση αυτή δεν είναι η μοναδική, υπάρχουν πολλές εναλλακτικές. Όμως δεν είναι εύκολο να γνωρίζουμε εκ των προτέρων ποια από τις πιθανές αναπαραστάσεις θ' αποδειχθεί βολικότερη. Ουσιαστικά, η επιλογή της κατάλληλης αναπαράστασης εξαρτάται από τον συγκεκριμένο τρόπο που το πρόγραμμά μας θα επεξεργάζεται τα δεδομένα. Εφόσον ακόμα δεν έχουμε γράψει το πρόγραμμα, μόνο η εμπειρία και η διαίσθηση μπορεί να μας καθοδηγήσει.

Κάτι Να "Μεταφράζει"

Εντάξει, επιλέξαμε την αναπαράσταση. Αλλά, όπως και να προχωρήσουμε, ο χρήστης θα πρέπει να βλέπει μια τρίλιζα για να παίζει, όχι εννέα τετράγωνα στη σειρά.

Στην Python, όπως και σε πολλές άλλες γλώσσες, κάθε είδους αρίθμηση ξεκινά από το `0`, όχι από το `1`. Έτσι, το πρώτο στοιχείο αυτής της λίστας που αναπαριστά τον πίνακα του παιχνιδιού βρίσκεται στη θέση `0` και το έναστο στη θέση `8`.

Γενικά, τα στοιχεία μιας λίστας μπορεί να είναι *ο,τιδήποτε*, ακόμα και άλλες λίστες.

Στη συγκεκριμένη περίπτωση, όλα τα στοιχεία της λίστας θα είναι αλφαριθμητικά και κάθε ένα από αυτά θα έχει τιμή είτε `"X"`, είτε `"O"`, είτε `" "` (κενό).

Ο τρόπος που το πρόγραμμα διαχειρίζεται "εσωτερικά" την αναπαράσταση της τρίλιζας δεν ενδιαφέρει το χρήστη. Στην οθόνη θα πρέπει να εμφανίζεται μια αναπαράσταση που να είναι γνώριμη στους παίκτες, δηλαδή ο κλασικός 3×3 πίνακας του παιχνιδιού.

Θα ξεκινήσουμε λοιπόν κατασκευάζοντας ένα υποπρόγραμμα που δέχεται σαν παράμετρο την εσωτερική αναπαράσταση `board` της τρίλιζας (εδώ πρόκειται για μια λίστα με εννέα στοιχεία) και την εμφανίζει στην οθόνη με τρόπο φιλικό προς το χρήστη.

```
def print3x3(board, trailing = True):
    """ Εμφανίζει σε διάταξη 3x3 τα περιεχόμενα
        μιας λίστας με 9 (τουλάχιστον) στοιχεία.
        board: Λίστα που θα εμφανιστεί σε διάταξη 3x3
    """
    print(" ", board[0], "|", board[1], "|", board[2])
    print(" ---+---+---")
    print(" ", board[3], "|", board[4], "|", board[5])
    print(" ---+---+---")
    print(" ", board[6], "|", board[7], "|", board[8])
    if trailing:
        print()
```

Μπορούμε ν' αναφερθούμε στα στοιχεία μιας λίστας με βάση τη *θέση* τους σε αυτή. Η αρίθμηση των θέσεων ξεκινά από το `0`. Έτσι, εδώ το πρώτο στοιχείο της λίστας `board` είναι το `board[0]`, το δεύτερο είναι το `board[1]`, κ.ο.κ.

Η παράμετρος `trailing` καθορίζει αν θα εμφανιστεί μια κενή γραμμή μετά τον 3×3 πίνακα και έχει σαν *προκαθορισμένη τιμή* την `True`. Αυτό σημαίνει ότι υπάρχει η δυνατότητα να μην καθοριστεί η τιμή της παραμέτρου κατά την κλήση της συνάρτησης. Στην περίπτωση αυτή, η παράμετρος θα πάρει την προκαθορισμένη της τιμή.

Αυτή η αντιστοιχία ανάμεσα στα τετράγωνα και τις θέσεις της λίστας, όπως εξάλλου και η ίδια η αναπαράσταση με λίστα, αφορά μόνο εμάς ως προγραμματιστές και ο χρήστης ούτε τη γνωρίζει, ούτε τον αφορά. Όταν το πρόγραμμά μας θα καλεί την `print3x3`, ο χρήστης θα βλέπει απλά στην οθόνη τα τετράγωνα της τρίλιζας.

Quick and Dirty

Για αρχή, νομίζω ότι μπορώ να φτιάξω κάτι στα γρήγορα που να λειτουργεί. Δεν θα είναι τέλειο και στην πορεία θ' αλλάξω αρκετά πράγματα, όμως έτσι θα διαπιστώσω ποια σημεία παρουσιάζουν δυσκολίες.

Αρχικές τιμές

Στο κύριο πρόγραμμα, η λίστα στην οποία θ' αποθηκεύουμε την τρέχουσα κατάσταση της τρίλιζας θα ονομάζεται `board`. Αρχικά, πριν ξεκινήσει το παιχνίδι, η τρίλιζα θα περιέχει εννέα κενά τετράγωνα.

```
# η αναπαράσταση της τρίλιζας:
# μια λίστα με 9 χαρακτήρες (" ", "X" ή "O")
# αρχικά, όλα τα τετράγωνα είναι κενά
board = 9 * [" "]
```

Όταν "πολλαπλασιάζουμε" μια λίστα με τον τελεστή * κατασκευάζουμε μια νέα λίστα που περιέχει πολλές φορές τα στοιχεία της αρχικής.

Εδώ, η λίστα `board` προκύπτει πολλαπλασιάζοντας επί `9` τη λίστα `[" "]`, που αποτελείται από ένα στοιχείο.

Όπως και στο Παιχνίδι της Αφαίρεσης, που ήταν επίσης παιχνίδι δύο παικτών, θα χρησιμοποιήσουμε μια μεταβλητή `player`, η οποία σε κάθε γύρο θα παίρνει *εναλλάξ* την τιμή `"X"` ή `"O"`, υποδεικνύοντας με αυτόν τον τρόπο ποιος έχει σειρά να παίξει σε κάθε γύρο. Κατά σύμβαση, πρώτος παίζει ο παίκτης με τα X, οπότε πριν ξεκινήσει το παιχνίδι η `player` θα πάρει την αντίστοιχη αρχική τιμή.

```
17 # ο παίκτης που θα παίξει πρώτος
18 player = "X"
```

Σχήμα 5.2: Η τιμή της μεταβλητής `player` θα εναλλάσσεται σε κάθε γύρο μεταξύ του `"X"` και του `"O"`, υποδεικνύοντας το σύμβολο του παίκτη που έχει σειρά να παίξει. Η αρχική της τιμή είναι το `"X"`.

Επανάληψη: η συνθήκη συνέχειας

Ας περάσουμε τώρα στην επαναληπτική δομή. Κάθε κύκλος της επανάληψης αντιστοιχεί στη συμπλήρωση ενός τετραγώνου από κάποιο παίκτη. Για να ολοκληρωθεί το παιχνίδι και να *τερματιστεί* η επανάληψη θα πρέπει είτε να συμπληρωθούν και τα εννέα τετράγωνα, είτε κάποιος από τους δύο παίκτες να κάνει τρίλιζα. Για να διατυπωθεί λοιπόν η συνθήκη της επανάληψης χρειάζεται το πρόγραμμα να καταμετρά το πλήθος των κενών τετραγώνων και να καταγράφει αν έχει γίνει τρίλιζα.

Για την καταμέτρηση των κενών τετραγώνων θα χρησιμοποιήσουμε τη μεταβλητή `blank`. Όταν ξεκινά το παιχνίδι, όλα τα τετράγωνα είναι κενά.

```
19 # πλήθος τετραγώνων που απομένουν κενά
20 blank = 9
```

Για να καταγράφει το πρόγραμμά μας αν έχει γίνει τρίλιζα ή όχι θα χρησιμοποιήσουμε μια λογική μεταβλητή `inarow`. Γνωρίζουμε βέβαια πως όταν ξεκινά το παιχνίδι, κανείς από τους δύο παίκτες δεν έχει κάνει τρίλιζα κι έτσι η αρχική τιμή της `inarow` θα πρέπει να είναι **`False`**.

```
21 # λογική μεταβλητή που δείχνει αν έχει γίνει τρίλιζα
22 inarow = False
```

Είμαστε τώρα έτοιμοι να διατυπώσουμε τη συνθήκη της επανάληψης: για να *συνεχιστεί* η επανάληψη θα πρέπει να απομένει τουλάχιστον ένα κενό τετράγωνο και ταυτόχρονα κανένας από τους δύο παίκτες να μην έχει κάνει τρίλιζα.

```
23 # επανάληψη: συνεχίζεται όσο υπάρχουν κενά τετράγωνα
24 #   και δεν έχει γίνει τρίλιζα
25 while blank > 0 and not inarow:
```

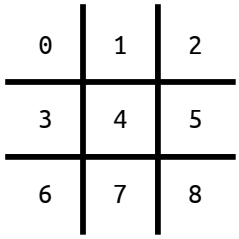
Επανάληψη: επιλογή κίνησης από τον παίκτη

Μέσα στην επανάληψη, στην αρχή κάθε νέου κύκλου, το πρόγραμμα θα εμφανίζει τον πίνακα της τρίλιζας στον παίκτη που έχει σειρά να παίξει και θα τον ρωτάει σε ποιο τετράγωνο επιθυμεί να παίξει.

Εδώ υπάρχει ένα πολύ λεπτό σημείο. Έστω ότι ζητάμε από το χρήστη να προσδιορίσει μ' έναν ακέραιο αριθμό το τετράγωνο στο οποίο θα παίξει. Υπονοείται λοιπόν ότι υπάρχει μια *αρίθμηση* των τετραγώνων της τρίλιζας, με βάση την οποία θα κάνει την επιλογή του ο παίκτης. Έχει μεγάλη σημασία ότι η αυτή η αρίθμηση των τετραγώνων αφορά την οπτική γωνία *του παίκτη*, τον τρόπο με τον οποίο θα

προσδιορίσει το τετράγωνο που τον ενδιαφέρει και *δεν έχει απαραίτητα σχέση* με την εσωτερική αναπαράσταση της τρίλιζας.

Εμείς όμως προς το παρόν θα επιλέξουμε την απλούστερη δυνατή αρίθμηση, η οποία *ταυτίζεται* με την αρίθμηση της εσωτερικής αναπαράστασης και φαίνεται στο σχήμα 5.3. Ας έχουμε όμως υπόψη ότι αυτή η αρίθμηση είναι βολική για εμάς *ως προγραμματιστές* και δεν είναι απαραίτητα η βολικότερη αρίθμηση *για το χρήστη*.



Σχήμα 5.3: Η αρίθμηση των τετραγώνων της τρίλιζας, από την σκοπιά του παίκτη. Με βάση αυτή την αρίθμηση επιλέγει ο παίκτης το τετράγωνο στο οποίο θα παίξει κάθε φορά. Θα μπορούσαμε να έχουμε επιλέξει μια διαφορετική αρίθμηση, αυτή όμως είναι βολική γιατί ταυτίζεται με την εσωτερική μας αναπαράσταση.

Στον κώδικα που ακολουθεί, η `print3x3` δεν χρησιμοποιείται μόνο για να εμφανιστεί ο πίνακας της τρίλιζας, αλλά επιστρατεύεται και μια δεύτερη φορά για να εμφανιστεί σε μορφή 3×3 πίνακα, σαν βοηθητικό υπόμνημα, η αρίθμηση με βάση την οποία θα επιλέξει τετράγωνο ο παίκτης.

```
26      # εμφάνιση πίνακα τρίλιζας και πιθανών κινήσεων
27      print3x3(board)
28      print3x3(range(9))
29      # επιλογή θέσης από τον παίκτη
30      print(player, "διάλεξε τετράγωνο:", end=" ")
31      position = int(input())
```

Η `range` είναι μια *ακολουθία* τιμών που ανήκουν σε ένα συγκεκριμένο *διάστημα*. Όταν η `range` καλείται με μια παράμετρο `n`, τότε το διάστημα τιμών είναι από το `0` μέχρι και το `n-1`.

Εδώ η `range(9)` είναι η *ακολουθία* των τιμών από το `0` μέχρι και το `8`, αντιστοιχεί δηλαδή στους αριθμούς των θέσεων της τρίλιζας.

Όταν ο χρήστης επιλέξει αριθμό τετραγώνου, το πρόγραμμα θα συμπληρώνει την αναπαράσταση του πίνακα της τρίλιζας με το σύμβολο του παίκτη, ενώ παράλληλα θα μειώνει το πλήθος των κενών τετραγώνων.

```
32      # ανακοίνωση κίνησης
33      print("Ο παίκτης", player, "παίζει στο", position)
34      # συμπλήρωση επιλεγμένης θέσης
35      board[position] = player
36      # μείωση κενών τετραγώνων κατά 1
37      blank -= 1
```

Χρησιμοποιώντας μια εντολή όπως η `board[position] = player` *τροποποιείται* το στοιχείο της λίστας `board` που βρίσκεται στη θέση `position`, το οποίο αντιστοιχεί πλέον σε μια νέα τιμή, την `player`. Είναι ένα βασικό χαρακτηριστικό των λιστών ότι τα στοιχεία τους μπορούν να τροποποιηθούν.

Η εντολή `blank -= 1` μειώνει την τιμή της μεταβλητής `blank` κατά μία μονάδα. Είναι ισοδύναμη με την εντολή `blank = blank - 1`. Ανάλογες μορφές μεταβολής μιας μεταβλητής είναι διαθέσιμες για την αύξηση, τον πολλαπλασιασμό και γενικότερα όλες τις αριθμητικές πράξεις.

Επανάληψη: έλεγχος για τρίλιζα

Μετά την κίνηση του παίκτη, το πρόγραμμά μας θα πρέπει να ελέγχει μήπως έγινε τρίλιζα. Προς το παρόν, θα υλοποιήσουμε αυτόν τον έλεγχο με τον πιο απλό τρόπο που μπορούμε να σκεφτούμε, εξετάζοντας όλες τις πιθανές τριάδες τετραγώνων. Στη συνέχεια θα έχουμε την ευκαιρία να διαπιστώσουμε αν υπάρχουν περιθώρια βελτίωσης.

```
38      # έλεγχος για τρίλιζα:
39      # για κάθε 3άδα θέσεων στις πιθανές τρίλιζες
40      # ελέγχεται αν ο παίκτης έχει καταλάβει 3 θέσεις
41      if board[0] == board[1] == board[2] == player:
42          inarow = True
43      elif board[3] == board[4] == board[5] == player:
44          inarow = True
45      elif board[6] == board[7] == board[8] == player:
46          inarow = True
```

```
47      elif board[0] == board[3] == board[6] == player:
48          inarow = True
49      elif board[1] == board[4] == board[7] == player:
50          inarow = True
51      elif board[2] == board[5] == board[8] == player:
52          inarow = True
53      elif board[0] == board[4] == board[8] == player:
54          inarow = True
55      elif board[2] == board[4] == board[6] == player:
56          inarow = True
```

Μέχρι στιγμής έχουμε δει πως οι συγκριτικοί τελεστές χρησιμοποιούνται για να συγκρίνουν δύο τιμές μεταξύ τους. Στην Python μπορούν να χρησιμοποιηθούν για να συγκριθούν *πολλές* τιμές μεταξύ τους, όπως σε αυτό το παράδειγμα με τον τελεστή ==. Αυτό είναι ένα χαρακτηριστικό που δεν το συναντάμε συχνά σε άλλες γλώσσες προγραμματισμού, με τους αντίστοιχους συγκριτικούς τελεστές.

Κάθε μία από τις οκτώ περιπτώσεις σε αυτή τη δομή επιλογής αντιστοιχεί σε έναν από τους οκτώ διαφορετικούς τρόπους να γίνει τρίλιζα. Και στις οκτώ περιπτώσεις εκτελείται η ίδια εντολή: η μεταβλητή `inarow` παίρνει την τιμή **`True`**, για να καταγραφεί πλέον από το πρόγραμμα ότι έχει γίνει τρίλιζα (εναλλακτικά, θα μπορούσε να χρησιμοποιηθεί μια *σύζευξη* των οκτώ επιμέρους συνθηκών). Παρατηρήστε ότι στη δομή επιλογής δεν υπάρχει **`else`** επειδή δε χρειάζεται να εκτελεστεί κάποια εντολή σε περίπτωση που δε γίνει τρίλιζα.

Επανάληψη: εναλλαγή παίκτη

Μέσα στην επανάληψη, απομένει μόνο να εναλλάσσουμε την τιμή της μεταβλητής `player`.

```
57      # εναλλαγή παίκτη
58      if player == "X":
59          player = "O"
60      else:
61          player = "X"
```

Ανακοίνωση αποτελέσματος

Όταν η επανάληψη τερματιστεί, το παιχνίδι θα έχει τελειώσει και το πρόγραμμα θα πρέπει να εμφανίσει στους παίκτες ποιο ήταν το αποτέλεσμα. Υπάρχουν δύο πιθανοί λόγοι για να τελειώσει το παιχνίδι: να έχει συμπληρωθεί ο πίνακας του παιχνιδιού ή να έχει γίνει τρίλιζα (χωρίς το ένα να αποκλείει το άλλο). Αν έχει γίνει τρίλιζα, θα πρέπει να εμφανίζεται κατάλληλο μήνυμα, διαφορετικά θα πρέπει να εμφανίζεται μήνυμα ότι το παιχνίδι έληξε ισόπαλο.

```
62  # εμφάνιση τελικού πίνακα τρίλιζας
63  print3x3(board)
64  # εμφάνιση αποτελέσματος
65  if inarow:
66      print("Τρίλιζα!")
67  else:
68      print("Ισοπαλία.")
```

oxo/src/oxo.1.py

Είναι *υποχρεωτικό* το πρόγραμμα να ελέγχει την τιμή της `inarow` και όχι της `blank` για να διαπιστώσει το αποτέλεσμα του παιχνιδιού. Η τιμή της `blank` *δεν* μπορεί να μας οδηγήσει σε ασφαλή συμπεράσματα, γιατί όταν έχει την τιμή `0` μετά το τέλος του παιχνιδιού, δεν ξέρουμε με βεβαιότητα αν το παιχνίδι έληξε ισόπαλο ή αν η τελευταία κίνηση οδήγησε σε τρίλιζα.

Συμμάζεμα

Ξέρω ότι μπορώ να "σπάσω" το πρόγραμμά μου σε μικρότερα τμήματα, υλοποιώντας τις επιμέρους λειτουργίες του προγράμματος ως ξεχωριστά υποπρογράμματα. Από που να ξεκινήσω;

Είναι σημαντικό να διακρίνουμε (σε οποιοδήποτε πρόγραμμα) τις ομάδες εντολών που λειτουργούν ως ενιαίο σύνολο και υλοποιούν μια συγκεκριμένη λειτουργία. Αυτά τα τμήματα του προγράμματος θα αποτελέσουν τη βάση για την κατασκευή των επιμέρους υποπρογραμμάτων.

Ο τρόπος με τον οποίο μπορούμε να διαιρέσουμε ένα ενιαίο πρόγραμμα σε επιμέρους τμήματα με βάση τη λειτουργία τους δεν είναι μοναδικός. Στην πραγματικότητα υπάρχουν πολλές εναλλακτικές. Επίσης, οι εμπειρότεροι προγραμματιστές συνήθως *σχεδιάζουν εκ των προτέρων* τα προγράμματά τους και τα υποπρογράμματα από τα οποία αποτελούνται, χωρίς αυτό να σημαίνει ότι στη συνέχεια, καθώς υλοποιούν το πρόγραμμά τους, δεν μπορούν να αναθεωρήσουν ή να εκλεπτύνουν τον αρχικό τους σχεδιασμό.

Αλληλεπίδραση με το χρήστη

Διατρέχοντας το πρόγραμμα που έχουμε αναπτύξει μέχρι στιγμής, εντοπίζουμε μέσα στην επαναληπτική δομή μια ομάδα εντολών που αναλαμβάνει την αλληλεπίδραση με τον παίκτη: εμφανίζει τα τετράγωνα της τρίλιζας και ρωτά τον παίκτη σε ποιο τετράγωνο επιθυμεί να παίξει.

```
# εμφάνιση πίνακα τρίλιζας και πιθανών κινήσεων
print3x3(board)
print3x3(range(9))
# επιλογή θέσης από τον παίκτη
print(player, "διάλεξε τετράγωνο:", end=" ")
position = int(input())
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `readPosition`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που έχει σειρά να παίξει και την αναπαράσταση `board` της τρίλιζας και επιστρέφει τη θέση που επέλεξε ο παίκτης για την επόμενη κίνησή του.

```
13 def readPosition(player, board):
14     """ Διαβάζει από τον παίκτη τη θέση
15         στην οποία επιθυμεί να παίξει και την
16         επιστρέφει.
17         player: Ο παίκτης που έχει σειρά να παίξει
18         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
19     """
20     # εμφάνιση πίνακα τρίλιζας και πιθανών κινήσεων
21     print3x3(board)
22     print3x3(range(9))
23     # επιλογή θέσης από τον παίκτη
24     print(player, "διάλεξε τετράγωνο:", end=" ")
25     position = int(input())
26     # επιστροφή επιλεγμένης θέσης
27     return position
```

Πραγματοποίηση κίνησης στο επιλεγμένο τετράγωνο

Αμέσως μετά τις εντολές που ζητούν από τον παίκτη να επιλέξει το τετράγωνο στο οποίο θα παίξει, υπάρχει μια ομάδα εντολών που πραγματοποιούν την κίνηση του παίκτη.

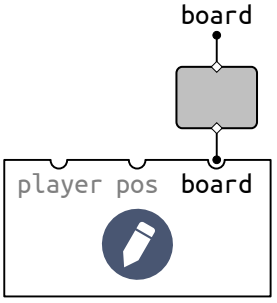
```
    position = int(input())
    # ανακοίνωση κίνησης
    print("Ο παίκτης", player, "παίζει στο", position)
    # συμπλήρωση επιλεγμένης θέσης
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `play`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που επέλεξε θέση, την αναπαράσταση `board` της τρίλιζας και τη θέση που επέλεξε ο παίκτης και πραγματοποιεί την κίνησή του.

```
28 def play(player, pos, board):
29     """ Πραγματοποιεί και ανακοινώνει την κίνηση
30         ενός παίκτη σε συγκεκριμένη θέση
31         player: Το σύμβολο του παίκτη
32         pos: Η θέση στην οποία παίζει ο παίκτης
33         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
34     """
35     # ανακοίνωση κίνησης
36     print("Ο παίκτης", player, "παίζει στο", pos)
37     # συμπλήρωση επιλεγμένης θέσης
38     board[pos] = player
```

Η συνάρτηση `play` δεν επιστρέφει κάποια τιμή, όμως *μεταβάλλει την αναπαράσταση της τρίλιζας* (σχήμα 5.4).

Σε αυτή, αλλά και σε όλες τις συναρτήσεις που ακολουθούν, ονομάσαμε `player` την παράμετρο που αντιστοιχεί στο σύμβολο του παίκτη και `board` την παράμετρο που αντιστοιχεί στην αναπαράσταση της τρίλιζας. Χρησιμοποιήσαμε δηλαδή τα ίδια ονόματα που χρησιμοποιούνται και στο κύριο πρόγραμμα. Αυτό δεν είναι υποχρεωτικό, θα μπορούσαμε να χρησιμοποιήσουμε άλλα ονόματα για τις παραμέτρους, απλά θεωρούμε ότι έτσι το πρόγραμμα είναι πιο κατανοητό. Πρέπει όμως να γίνει σαφές ότι *δεν πρόκειται για τις ίδιες μεταβλητές*: η `player` και η `board` του κύριου προγράμματος δεν ταυτίζονται με τις *τοπικές* `player` και `board` μέσα σε μια συνάρτηση. Μάλιστα οι τοπικές μεταβλητές των συναρτήσεων δημιουργούνται μόνο όταν καλείται η συνάρτηση και παύουν να υφίστανται όταν ολοκληρωθεί η εκτέλεση των εντολών της.



Σχήμα 5.4: Όταν κληθεί η συνάρτηση `play` από το κύριο πρόγραμμα, θα χρησιμοποιηθεί σαν παράμετρος η λίστα `board` του κύριου προγράμματος. Η τοπική μεταβλητή `board` της συνάρτησης `play` και η `board` του κύριου προγράμματος είναι δύο διαφορετικές μεταβλητές. Ωστόσο, είναι στην πραγματικότητα δύο διαφορετικά ονόματα για το ίδιο πράγμα: τη λίστα που αποτελεί την αναπαράσταση της τρίλιζας. Έτσι, οποιαδήποτε τροποποίηση γίνει εντός της συνάρτησης στα στοιχεία της λίστας `board`, αφορά ουσιαστικά και τη λίστα `board` του κύριου προγράμματος.

Έλεγχος για τρίλιζα

Μια άλλη ομάδα εντολών που αποτελούν ενιαίο σύνολο απαρτίζεται από τη δομή επιλογής που ελέγχει τις πιθανές τριάδες τετραγώνων για να διαπιστωθεί αν ο παίκτης που έπαιξε έκανε τρίλιζα.

```
# έλεγχος για τρίλιζα:
# για κάθε 3άδα θέσεων στις πιθανές τρίλιζες
# ελέγχεται αν ο παίκτης έχει καταλάβει 3 θέσεις
if board[0] == board[1] == board[2] == player:
    inarow = True
elif board[3] == board[4] == board[5] == player:
    inarow = True
...
    inarow = True
elif board[2] == board[4] == board[6] == player:
    inarow = True
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `check`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που έχει σειρά να παίξει και την αναπαράσταση `board` της τρίλιζας κι επιστρέφει **`True`** ή **`False`** ανάλογα αν ο παίκτης έκανε τρίλιζα ή όχι.

```
39 def check(player, board):
40     """ Ελέγχει αν υπάρχει 3άδα όπου ο παίκτης player
41         έχει καταλάβει και τις 3 θέσεις.
42         player: σύμβολο παίκτη ("X" ή "O")
43         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
44     """
45     # για κάθε 3άδα θέσεων στις πιθανές τρίλιζες
46     # αν ο παίκτης έχει καταλάβει και τις 3 θέσεις
47     if board[0] == board[1] == board[2] == player:
48         return True
49     elif board[3] == board[4] == board[5] == player:
50         return True
51     elif board[6] == board[7] == board[8] == player:
52         return True
53     elif board[0] == board[3] == board[6] == player:
54         return True
55     elif board[1] == board[4] == board[7] == player:
56         return True
57     elif board[2] == board[5] == board[8] == player:
58         return True
59     elif board[0] == board[4] == board[8] == player:
60         return True
61     elif board[2] == board[4] == board[6] == player:
62         return True
63     else:
64         # αν φτάσουμε μέχρι εδώ, ο έλεγχος απέτυχε
65         return False
```

Εναλλαγή παίκτη

Τέλος, οι εντολές που φροντίζουν για την εναλλαγή του παίκτη στο τέλος κάθε κύκλου της επανάληψης αποτελούν επίσης ένα ενιαίο σύνολο εντολών.

```
    # εναλλαγή παίκτη
    if player == "X":
        player = "O"
    else:
        player = "X"
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `next`, η οποία δέχεται σαν παράμετρο έναν παίκτη `player` και επιστρέφει τον παίκτη που έχει σειρά να παίξει μετά από αυτόν.

```
66 def next(player):
67     """ Επιστρέφει το σύμβολο του παίκτη που παίζει
68         μετά τον παίκτη με σύμβολο player.
69         player: σύμβολο παίκτη ("X" ή "O")
70     """
71     if player == "X":
72         return "O"
73     else:
74         return "X"
```

Το κύριο πρόγραμμα

Τώρα, το κύριο πρόγραμμα μπορεί να καλεί αυτές τις συναρτήσεις. Στο σημείο όπου το πρόγραμμα αλληλεπιδρά με το χρήστη για να τον ρωτήσει σε ποιο τετράγωνο επιθυμεί να παίξει, καλείται η συνάρτηση `readPosition`. Στο σημείο όπου πραγματοποιείται η κίνηση του παίκτη, καλείται η `play`. Στο σημείο όπου ελέγχεται αν η κίνηση του παίκτη οδηγεί σε τρίλιζα, καλείται η `check`. Τέλος, αμέσως μετά, στο σημείο όπου εναλλάσσεται ο παίκτης που έχει σειρά να παίξει, καλείται η `next`.

```
85 # επανάληψη: συνεχίζεται όσο υπάρχουν κενά τετράγωνα
86 #   και δεν έχει γίνει τρίλιζα
87 while blank > 0 and not inarow:
88     # επιλογή θέσης από τον παίκτη
89     position = readPosition(player, board)
90     # συμπλήρωση επιλεγμένης θέσης
91     play(player, position, board)
92     # μείωση κενών τετραγώνων κατά 1
93     blank -= 1
94     # έλεγχος για τρίλιζα
95     inarow = check(player, board)
96     # εναλλαγή παίκτη
97     player = next(player)
```

Τώρα πλέον το μέγεθος του κύριου προγράμματος έχει μειωθεί δραματικά, ενώ έχει βελτιωθεί κατά πολύ η αναγνωσιμότητά του.

Το Ζήτημα του Νικητή

Θα ήθελα το πρόγραμμα ν' ανακοινώνει ποιος παίκτης κέρδισε.

Σε περίπτωση που γίνει τρίλιζα, το πρόγραμμα εμφανίζει το μήνυμα `"Τρίλιζα!"`, χωρίς όμως ν' ανακοινώνει ποιος από τους δύο παίκτες κέρδισε. Προφανώς, πρόκειται για τον παίκτη που έπαιξε τελευταίος κι έτσι μια βιαστική λύση θα ήταν να εμφανίσουμε το νικητή τροποποιώντας τις εντολές εμφανίζουν το αποτέλεσμα ως εξής:

```
# εμφάνιση αποτελέσματος
if inarow:
    print("Τρίλιζα! Κέρδισε ο", player)
else:
    print("Ισοπαλία.")
```

Εδώ όμως υπάρχει πρόβλημα. Η μεταβλητή `player` εναλλάσσεται *στο τέλος* κάθε κύκλου της επανάληψης, ακόμα κι όταν κάποιος παίκτης κάνει τρίλιζα. Έτσι, μετά τον τερματισμό της επανάληψης, η `player` δεν αντιστοιχεί στον παίκτη που έπαιξε τελευταίος (και κέρδισε), αλλά στον επόμενο (που είναι ο ηττημένος). Επομένως, αν το πρόγραμμα απλά εμφανίσει την τιμή της `player`, θα εμφανίσει τον παίκτη που *έχασε*, όχι εκείνον που έκανε τρίλιζα.

Για να διορθώσουμε αυτή την συμπεριφορά θα υιοθετήσουμε μια λύση που απαιτεί τις λιγότερες αλλαγές στο πρόγραμμά μας: ο νικητής του παιχνιδιού είναι ο παίκτης που θα έπαιζε μετά τον ηττημένο.

```
100 # εμφάνιση αποτελέσματος
101 if inarow:
102     print("Τρίλιζα! Κέρδισε ο", next(player))
103 else:
104     print("Ισοπαλία.")
```

oxo/src/oxo.2.py

Το Ζήτημα του Ελέγχου

Όταν ο χρήστης επιλέγει το τετράγωνο στο οποίο θα παίξει, το πρόγραμμα του επιτρέπει να πληκτρολογήσει οποιονδήποτε αριθμό, χωρίς κανέναν έλεγχο αν αυτός βρίσκεται μεταξύ `0` *και* `8`*. Δεν ελέγχεται καν αν το τετράγωνο που επιλέγει ο χρήστης είναι κατειλημμένο.*

Η επιλογή τετραγώνου από τον παίκτη πραγματοποιείται μέσω της συνάρτησης `readPosition`. Αυτή θα πρέπει τώρα να *επεκταθεί*, έτσι ώστε η τιμή που πληκτρολογεί ο παίκτης να ελέγχεται και, σε περίπτωση που δεν είναι έγκυρη, να ζητείται νέα τιμή.

Μέσα στη `readPosition`, οι εντολές που ζητούν από τον παίκτη να επιλέξει το τετράγωνο στο οποίο θα παίξει θα τοποθετηθούν πλέον μέσα σε μια επαναληπτική δομή. Έτσι, ο παίκτης θα "εγκλωβίζεται"

σε έναν κύκλο από τον οποίο βγαίνει μόνο όταν πληκτρολογήσει μια έγκυρη τιμή. Αν ο παίκτης επιλέξει θέση που δεν είναι ανάμεσα στο `0` και το `8` ή επιλέξει κατειλημμένη θέση τότε εμφανίζεται μήνυμα λάθους και η ανάγνωση τιμής επαναλαμβάνεται.

```
20      # εμφάνιση πίνακα τρίλιζας και πιθανών κινήσεων
21      print3x3(board)
22      print3x3(range(9))
23      # επανάληψη: όσο το τετράγωνο δεν είναι έγκυρο
24      while True:
25          # επιλογή θέσης από τον παίκτη
26          print(player, "διάλεξε τετράγωνο:", end=" ")
27          position = int(input())
28          # έλεγχος εγκυρότητας
29          if position < 0 or position > 8:
30              print("Επίλεξε τιμή μεταξύ 0 και 8.")
31          elif board[position] != " ":
32              print("Το", position, "δεν είναι κενό.")
33          else:
34              # έγκυρη τιμή, τέλος επανάληψης
35              break
36      # επιστροφή επιλεγμένης θέσης
37      return position
```

oxo/src/oxo.3.py

Στο κύριο πρόγραμμα δεν απαιτείται καμία αλλαγή. Εκεί εξακολουθεί να καλείται η `readPosition`, από την οποία προκύπτει η θέση στην οποία θα τοποθετήσει το σύμβολό του ο παίκτης. Όμως τώρα η τιμή που θα επιστρέψει η τροποποιημένη `readPosition` θα είναι σίγουρα έγκυρη.

Το Ζήτημα της Τρίλιζας

Η συνάρτηση `check` *μου κάθεται στο λαιμό. Δεν γίνεται να υλοποιήσουμε τον έλεγχο αν έχει γίνει τρίλιζα με πιο κομψό, συμπαγή τρόπο;*

Αν μελετήσουμε τη μεγάλη **`if`** που βρίσκεται μέσα στη συνάρτηση `check` και ελέγχει τους οκτώ διαφορετικούς τρόπους να γίνει τρίλιζα, θα παρατηρήσουμε ότι οι οκτώ συνθήκες που ελέγχονται είναι ουσιαστικά ίδιες μεταξύ τους. Κάθε συνθήκη εξετάζει μια τριάδα θέσεων και το μόνο που αλλάζει από περίπτωση σε περίπτωση είναι *οι αριθμοί των τριών θέσεων που ελέγχονται.*

Ας καταγράψουμε λοιπόν σε μια κατάλληλη δομή τις διαφορετικές τριάδες θέσεων που εξετάζονται.

```
# οι 3άδες των θέσεων που σχηματίζουν τρίλιζες
triples = (
    (0,1,2),(3,4,5),(6,7,8),    # οριζόντια
    (0,3,6),(1,4,7),(2,5,8),    # κάθετα
    (0,4,8),(2,4,6))            # διαγώνια
```

Μια δομή δεδομένων που μοιάζει πολύ με τις λίστες είναι οι *πλειάδες* (tuples). Τα περιεχόμενα των πλειάδων περικλείονται σε παρενθέσεις και, σε αντίθεση με τις λίστες, *δεν* μπορούν να μεταβληθούν. Η πρόσβαση στα στοιχεία των πλειάδων γίνεται ακριβώς με τον ίδιο τρόπο όπως και στις λίστες.

Εδώ θέλουμε ν' αποτυπώσουμε τις διαφορετικές τριάδες θέσεων που σχηματίζουν τρίλιζες. Επειδή αυτές οι τιμές δεν πρόκειται ν' αλλάξουν, χρησιμοποιείται για την αποθήκευσή τους μια πλειάδα, η `triples`.

Η `triples` είναι μια πλειάδα που περιέχει άλλες πλειάδες. Συγκεκριμένα, περιέχει τριάδες με τους αριθμούς των θέσεων που σχηματίζουν τρίλιζες.

Αυτό το τμήμα κώδικα που ορίζει την `triples`, μπορούμε να το εισάγουμε στη συνάρτηση `check`. Στην περίπτωση αυτή, θα πρόκειται για μια *τοπική* μεταβλητή της συνάρτησης. Θα δημιουργείται εκ νέου κάθε φορά που θα καλείται η `check` και θα παύει να υφίσταται όταν τελειώνει η εκτέλεσή της.

Εμείς όμως θα προτιμήσουμε να ορίσουμε την `triples` *στην αρχή του κύριου προγράμματος*. Θα είναι έτσι μια *καθολική* μεταβλητή. Θα μπορούν να αναφερθούν σε αυτήν (αλλά όχι να την τροποποιήσουν) όλες οι συναρτήσεις, συμπεριλαμβανομένης και της `check` φυσικά.

Διατρέχοντας τις οκτώ τριάδες που περιέχει η `triples`, μπορούμε να ελέγξουμε *επαναληπτικά*, για κάθε μια από αυτές, αν είναι πλήρως συμπληρωμένη από έναν συγκεκριμένο παίκτη. Η συνάρτηση `check` μπορεί πλέον να ξαναγραφτεί με πιο συμπαγή τρόπο.

```
49 def check(player, board):
50     """ Ελέγχει αν υπάρχει 3άδα όπου ο παίκτης player
51         έχει καταλάβει και τις 3 θέσεις.
52         player: σύμβολο παίκτη ("X" ή "O")
53         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
54     """
55     # για κάθε 3άδα θέσεων
56     for triple in triples:
57         # p1, p2 και p3 οι τρεις θέσεις της 3άδας
58         p1, p2, p3 = triple
59         # αν ο παίκτης έχει καταλάβει και τις 3 θέσεις
60         if board[p1] == board[p2] == board[p3] == player:
61             return True
62     # αν φτάσουμε μέχρι εδώ, ο έλεγχος απέτυχε
63     return False
```

oxo/src/oxo.4.py

Η εντολή **for** είναι μια εντολή επανάληψης που διατρέχει τα στοιχεία μιας ακολουθίας τιμών, όπως μια λίστα ή μια πλειάδα, με τη σειρά με την οποία αυτά εμφανίζονται στην ακολουθία. Σε κάθε επανάληψη, η τιμή του επόμενου στοιχείου της ακολουθίας ανατίθεται στη *μεταβλητή απαρίθμησης* που χρησιμοποιούμε στη **for**.

Εδώ η **for** χρησιμοποιείται για να διατρεχθούν όλες οι τριάδες της πλειάδας `triples`.

Με την εντολή `p1, p2, p3 = triple` οι τρεις τιμές που βρίσκονται αποθηκευμένες στην πλειάδα `triple` αποδίδονται στις μεταβλητές `p1`, `p2` και `p3` αντίστοιχα.

Αυτό ονομάζεται *ξεπακετάρισμα* της πλειάδας (tuple unpacking) και είναι ουσιαστικά ο μηχανισμός που χρησιμοποιείται και για την επιστροφή πολλαπλών τιμών από συναρτήσεις.

Αν και η συνάρτηση `check` ουσιαστικά ξαναγράφτηκε από την αρχή, η *λειτουργία* της παρέμεινε η ίδια: ελέγχει αν έχει γίνει τρίλιζα. Γι' αυτό και δεν απαιτείται καμία τροποποίηση στο κύριο πρόγραμμα. Αυτό που άλλαξε είναι η *υλοποίηση* αυτής της λειτουργίας, δηλαδή ο τρόπος με τον οποίο ελέγχεται αν έχει γίνει τρίλιζα.

Κάτι Για Παρέα

Στην αρχή είπαμε ότι θα φτιάξουμε ένα πρόγραμμα που θα παίζει τρίλιζα. Προς το παρόν και οι δύο παίκτες που συμμετέχουν στο παιχνίδι είναι άνθρωποι.

Θα κάνουμε τις απαραίτητες επεκτάσεις, ώστε το πρόγραμμα ν' αναλάβει το ρόλο ενός από τους δύο παίκτες. Δεν είναι ανάγκη να προσπαθήσουμε εξαρχής να κάνουμε το πρόγραμμα να παίζει έξυπνα. Με αυτό θα ασχοληθούμε στο επόμενο βήμα· προς το παρόν, θα του επιτρέψουμε να ξεκινάει πρώτο και να επιλέγει σε κάθε γύρο μια τυχαία κίνηση.

Αφού οι κινήσεις του προγράμματος θα είναι τυχαίες, χρειάζεται να εισαγάγουμε, στην αρχή του προγράμματος, την κατάλληλη βιβλιοθήκη.

```
1 import random
```

Οι διαθέσιμες θέσεις

Για να επιλέξει το πρόγραμμα σε ποια θέση θα παίξει, θα πρέπει πρώτα να υπολογίσει ποιες από τις εννέα θέσεις είναι διαθέσιμες.

Αυτό μπορεί να γίνει με τον κώδικα που ακολουθεί, ο οποίος διατρέχει τις εννέα θέσεις της τρίλιζας και κατασκευάζει μια λίστα με τις θέσεις εκείνες που είναι διαθέσιμες.

```
# η λίστα με τις διαθέσιμες θέσεις, αρχικά κενή
positions = []
# για κάθε θέση από 0 μέχρι και 8
for s in range(9):
    # αν η θέση s είναι διαθέσιμη
    if board[s] == " ":
        # προστίθεται στη λίστα διαθέσιμων θέσεων
        positions.append(s)
```

Η `range` χρησιμοποιείται για να αναφερθούμε σε ένα *διάστημα τιμών*. Η προαιρετική πρώτη παράμετρος ορίζει το αριστερό άκρο του διαστήματος. Αν παραληφθεί, το διάστημα ξεκινά από το *μηδέν*. Η δεύτερη παράμετρος ορίζει το δεξί άκρο, το οποίο είναι ανοικτό (δηλαδή το διάστημα δεν περιλαμβάνει την τιμή που δίνεται σαν δεξί άκρο). Η προαιρετική τρίτη παράμετρος ορίζει το *βήμα*, δηλαδή ανά πόσες τιμές του διαστήματος θα διατρέχονται.

Η μέθοδος `append()` εφαρμόζεται σε μια λίστα και προσθέτει ένα νέο στοιχείο στο τέλος της.

Είναι συνηθισμένο να κατασκευάζουμε λίστες με αυτόν ακριβώς τον τρόπο, ξεκινώντας από μια κενή λίστα στην οποία προσθέτουμε σταδιακά τις τιμές που ικανοποιούν ένα συγκεκριμένο κριτήριο. Είναι τόσο συνηθισμένο αυτό το μοτίβο, που υπάρχει κι ένας εναλλακτικός, συνοπτικότερος τρόπος να δημιουργηθεί η ίδια λίστα:

```
# συγκέντρωση λίστας: λίστα με τις διαθέσιμες θέσεις s
positions = [s for s in range(9) if board[s] == " "]
```

Εδώ η `append()` καλείται επαναληπτικά, κατασκευάζοντας στοιχείο προς στοιχείο τη λίστα με τις διαθέσιμες θέσεις.

Η *συγκέντρωση λίστας* (list comprehension), είναι ένας συνοπτικός και εύληπτος τρόπος να δημιουργήσουμε μια λίστα περιγράφοντας τα στοιχεία της, από που αυτά προέρχονται και πως επιλέγονται.

Ουσιαστικά εδώ *περιγράφουμε* από τι αποτελείται η `positions`, όπως θα κάναμε στα μαθηματικά: είναι μια λίστα από όλες τις θέσεις `s`, μεταξύ `0` και `8`, για τις οποίες ισχύει ότι το `board[s]` είναι κενό, δηλαδή η αντίστοιχη θέση της τρίλιζας είναι διαθέσιμη.

Εδώ συγκεντρώνουμε σε μια λίστα όλα τα στοιχεία `s` που περιέχονται στο διάστημα μεταξύ `0` και `8` και έχουν το χαρακτηριστικό ότι το `board[s]` έχει την τιμή `" "` (κενό).

Τώρα μπορούμε να ορίσουμε τη συνάρτηση `available`, η οποία δέχεται σαν παράμετρο την αναπαράσταση `board` της τρίλιζας και επιστρέφει *μια νέα λίστα* με τους αριθμούς των διαθέσιμων θέσεων.

```
79 def randomPosition(board):
80     """ Επιστρέφει μια τυχαία διαθέσιμη θέση
81         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
82     """
83     return random.choice(available(board))
```

Τυχαία επιλογή θέσης

Έχοντας κατασκευάσει τη συνάρτηση `available`, η οποία επιστρέφει μια λίστα με τις διαθέσιμες θέσεις, είναι εύκολο να προχωρή-

σουμε και στον ορισμό της συνάρτησης `randomPosition`, η οποία δέχεται σαν παράμετρο την αναπαράσταση `board` της τρίλιζας και επιστρέφει μια τυχαία επιλεγμένη διαθέσιμη θέση.

```
79 def randomPosition(board):
80     """ Επιστρέφει μια τυχαία διαθέσιμη θέση
81         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
82     """
83     return random.choice(available(board))
```

Η συνάρτηση `choice()` της βιβλιοθήκης `random` δέχεται σαν παράμετρο μια λίστα κι επιστρέφει ένα τυχαία επιλεγμένο στοιχείο της.

Εδώ χρησιμοποιείται για την τυχαία επιλογή μιας θέσης, από τη λίστα των διαθέσιμων θέσεων.

Επεκτάσεις στο κύριο πρόγραμμα

Ως τώρα, έχουμε προσθέσει στο πρόγραμμα την απαραίτητη υποδομή ώστε να είναι δυνατή η επιλογή μιας τυχαίας θέσης. Ωστόσο, το κύριο πρόγραμμα εξακολουθεί να υλοποιεί ένα παιχνίδι δύο παικτών. Θα πρέπει λοιπόν να επεκτείνουμε το κύριο πρόγραμμα, ώστε ν' αναλαμβάνει πλέον το ρόλο ενός από τους δύο παίκτες.

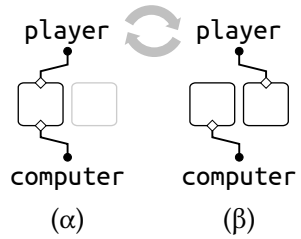
Θα χρησιμοποιήσουμε τη μεταβλητή `computer`, της οποίας η τιμή θ' αντιστοιχεί στο σύμβολο του παίκτη που θα παίζει αυτοματοποιημένα. Στην αρχή του παιχνιδιού η `computer` θα παίρνει την τιμή `"X"`, ώστε το πρόγραμμα να παίζει πρώτο.

```
95 # ο παίκτης που θα κατευθύνεται από το πρόγραμμα
96 computer = "X"
```

Θυμηθείτε ότι η μεταβλητή `player` εναλλάσσεται σε κάθε γύρο μεταξύ των τιμών `"X"` και `"O"`, υποδεικνύοντας ποιος παίκτης έχει σειρά να παίξει στον επόμενο γύρο. Συκρίνοντας λοιπόν την `player` με την `computer` ελέγχουμε ουσιαστικά αν ο επόμενος παίκτης είναι ο άνθρωπος ή το πρόγραμμά μας. Αν είναι σειρά του προγράμματος να παίξει, καλείται η `randomPosition`, ενώ σε διαφορετική περίπτωση καλείται η `readPosition`, ώστε να επιλέξει ο χρήστης τη θέση στην οποία επιθυμεί να παίξει.

```
104     if player == computer:
105         # επιλογή θέσης από το πρόγραμμα
106         position = randomPosition(board)
107     else:
108         # επιλογή θέσης από τον παίκτη
109         position = readPosition(player, board)
```

oxo/src/oxo.5.py



Σχήμα 5.5: Επιλογή κίνησης, ανάλογα με τον παίκτη: (α) αν η `computer` έχει ίδια τιμή με την `player`, τότε είναι σειρά του προγράμματος να παίξει, ενώ (β) σε διαφορετική περίπτωση, έχει σειρά να παίξει ο χρήστης.

Κάτι "Εξυπνότερο" Για Παρέα

Ωραία, τώρα το πρόγραμμμα παίζει με τον παίκτη, αλλά οι κινήσεις του είναι αστείες. Παίζει τυχαία ακόμα κι όταν του δίνεται η ευκαιρία να κερδίσει. Παίζει τυχαία ακόμα κι όταν ο αντίπαλός του ετοιμάζεται να κάνει τρίλιζα. Θέλω να παίζει "εξυπνότερα".

Το λιγότερο που μπορούμε να κάνουμε είναι να επεκτείνουμε το πρόγραμμα έτσι ώστε να ελέγχει πότε ένας παίκτης έχει *τη δυνατότητα*

να κάνει τρίλιζα. Έτσι, όταν έχει την ευκαιρία, το πρόγραμμα θα μπορεί να κερδίζει ή, τουλάχιστον, να εμποδίζει τον αντίπαλό του.

Ένας παίκτης έχει τη δυνατότητα να κάνει τρίλιζα όταν υπάρχει τριάδα στην οποία *οι δύο θέσεις* είναι ήδη κατειλημμένες από τον παίκτη και η τρίτη είναι κενή.

Η συνάρτηση `check` που περιέχει *ήδη* το πρόγραμμά μας ελέγχει κάτι παρεμφερές: αν υπάρχει τριάδα στην οποία *και οι τρεις θέσεις* είναι ήδη κατειλημμένες από έναν παίκτη.

Μπορούμε λοιπόν να βασιστούμε στον κώδικα της `check`, κάνοντας τις κατάλληλες τροποποιήσεις, και να φτιάξουμε τη νέα συνάρτηση `checkPartial`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που έχει σειρά να παίξει και την αναπαράσταση `board` της τρίλιζας και ελέγχει αν ο παίκτης έχει τη δυνατότητα να κάνει τρίλιζα. Η συνάρτηση επιστρέφει τον αριθμό της θέσης στην οποία πρέπει να παίξει ο παίκτης `player` για να κάνει τρίλιζα ή, αν δεν υπάρχει τέτοια θέση, την ειδική τιμή `None`.

```
65 def checkPartial(player, board):
66     """ Ελέγχει αν υπάρχει 3άδα όπου ο παίκτης player
67         έχει καταλάβει τις 2 από τις τρεις θέσεις.
68         Επιστρέφει την 3η κενή θέση της 3άδας ή None.
69         player: σύμβολο παίκτη ("X" ή "O")
70         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
71     """
72     # για κάθε 3άδα θέσεων
73     for triple in triples:
74         # p1, p2 και p3 οι τρεις θέσεις της 3άδας
75         p1, p2, p3 = triple
76         # αν ο παίκτης κατέχει 2 από τις 3 θέσεις
77         if (board[p1] == board[p2] == player and
78             board[p3] == " "):
79             return p3
80         elif (board[p1] == board[p3] == player and
81             board[p2] == " "):
82             return p2
83         elif (board[p2] == board[p3] == player and
84             board[p1] == " "):
85             return p1
86     # αν φτάσουμε μέχρι εδώ, ο έλεγχος απέτυχε
87     return None
```

Η χρήση των παρενθέσεων στις συνθήκες της `if` δεν είναι υποχρεωτική. Είναι όμως ένας από τους πιθανούς τρόπους να υποδηλώσουμε ότι πρόκειται για "μεγάλες" γραμμές που συνεχίζονται και στην επόμενη γραμμή. Ένας άλλος τρόπος είναι η χρήση της καθέτου \ στο τέλος κάθε τέτοιας μεγάλης γραμμής.

Η `checkPartial`, όπως και η `check`, διατρέχει τις τριάδες θέσεων που σχηματίζουν τρίλιζες. Όμως η `checkPartial` ελέγχει, για κάθε τριάδα, αν είναι *σχεδόν* συμπληρωμένη από έναν παίκτη και επιστρέφει τον αριθμό της θέσης που απομενει να συμπληρωθεί.

Τυχαιότητα με Μέτρο

Έχοντας στη διάθεσή μας την `checkPartial`, μπορούμε να επεκτείνουμε την `randomPosition` έτσι ώστε ο παίκτης με τα X να παίζει τυχαία μόνο όταν δεν έχει την ευκαιρία να νικήσει ή να εμποδίσει τον αντίπαλό του.

Επειδή αυτή η ευκαιρία παρουσιάζεται μετά τη δεύτερη κίνηση, η `randomPosition` θα ξεκινά πλέον με τον κώδικα που ακολουθεί:

```
108     # πόσες κινήσεις έχει κάνει ο X;
109     nbMoves = board.count("X")
```

Η μέθοδος `count` εφαρμόζεται σε μια λίστα. Δέχεται σαν παράμετρο μια τιμή και επιστρέφει το πλήθος των εμφανίσεων αυτής της τιμής στη λίστα.

Εδώ η `count` χρησιμοποιείται για να μετρήσουμε το πλήθος των `"X"` στη λίστα `board`, δηλαδή πόσες φορές έχει ήδη παίξει ο παίκτης με τα X.

Αν έχουν γίνει τουλάχιστον δύο κινήσεις, ελέγχεται αν υπάρχει θέση στην οποία μπορεί να παίξει ο παίκτης με τα X και να κάνει τρίλιζα. Αν υπάρχει, η `randomPosition` επιστρέφει αυτή τη θέση, αντί μιας τυχαίας, επιτρέποντας στον παίκτη με τα X να κερδίσει.

```
110     # αν ο "X" έχει κάνει περισσότερες από 2 κινήσεις
111     if nbMoves >= 2:
112         # έλεγξε αν ο "X" μπορεί να κάνει τρίλιζα
113         position = checkPartial("X", board)
114         if position is not None:
115             return position
```

Στη συνέχεια, ελέγχεται αν υπάρχει θέση στην οποία μπορεί να παίξει ο παίκτης με τα O και να κάνει τρίλιζα. Αν υπάρχει, επιστρέφεται αυτή η θέση, αντί μιας τυχαίας, επιτρέποντας στον παίκτη με τα X να εμποδίσει τον αντίπαλό του από το να κάνει τρίλιζα.

```
116         # έλεγξε αν ο "O" μπορεί να κάνει τρίλιζα
117         position = checkPartial("O", board)
118         if position is not None:
119             return position
120     # διαφορετικά επίλεξε τυχαία μια διαθέσιμη θέση
121     return random.choice(available(board))
```

oxo/src/oxo.6.py

Τώρα η `randomPosition` επιστρέφει μια τυχαία θέση μόνο αν δεν προκύψει αποτέλεσμα από την όλη διαδικασία που προηγείται.

Το Έξυπνο Χαρτί

Τίποτα καλύτερο δε γίνεται; Εγώ ξέρω ότι οι υπολογιστές είναι αχτύπητοι σε τέτοιου είδους παιχνίδια.

Συνήθως τα προγράμματα που παίζουν τέτοια παιχνίδια χρειάζεται να κάνουν *αναζήτηση* για να παίξουν έξυπνα. Αυτό σημαίνει ότι δοκιμάζουν πολλούς διαφορετικούς συνδυασμούς κινήσεων για να καταλήξουν στην επόμενη κίνηση που θα επιλέξουν. Όμως η τρίλιζα είναι πολύ απλό παιχνίδι και δε χρειάζεται αναζήτηση. Αρκούν ορισμένες απλές οδηγίες για να παίξει κανείς ικανοποιητικά.

Εμείς θα δανειστούμε τις οδηγίες που περιγράφονται σε μια διασκεδαστική δραστηριότητα που ονομάζεται *Το Έξυπνο Χαρτί*. Και πάλι, θεωρούμε ότι το πρόγραμμά μας θα παίξει πρώτο, χρησιμοποιώντας το σύμβολο Χ.

Έξυπνο χαρτί: `pythonies.mysch.gr/ipaper.pdf` και οι επεκτάσεις του: `ipaper-ext.pdf`

> *Κίνηση 1:* Γράψε το Χ σε οποιαδήποτε γωνία.
>
> *Κίνηση 2:* Αν η γωνία που βρίσκεται διαγωνίως απέναντι από το πρώτο Χ είναι ελεύθερη, τότε γράψε εκεί το Χ, αλλιώς γράψε το Χ σε οποιαδήποτε ελεύθερη γωνία.
>
> *Κινήσεις 3 και 4:* Αν μπορείς, κάνε τρίλιζα με τα Χ. Διαφορετικά, έλεγξε αν ο αντίπαλος μπορεί να κάνει τρίλιζα και γράψε το Χ έτσι ώστε να τον εμποδίσεις. Αν κανένας δεν μπορεί να κάνει τρίλιζα, γράψε το Χ σε οποιαδήποτε ελεύθερη γωνία.
>
> *Κίνηση 5:* Γράψε το Χ στο ελεύθερο τετράγωνο.

Η συνάρτηση `paperPosition` που ακολουθεί δέχεται σαν παράμετρο την αναπαράσταση `board` της τρίλιζας και επιστρέφει τη θέση στην οποία πρέπει να παίξει ο παίκτης με τα Χ, σύμφωνα με τις οδηγίες του έξυπνου χαρτιού.

Αυτή η συνάρτηση θ' αντικαταστήσει την `randomPosition` που επιλέγει τυχαίες κινήσεις.

```
100 def paperPosition(board):
101     """ Επιστρέφει τον αριθμό της θέσης όπου πρέπει
102         να παίξει ο Χ, σύμφωνα με το "έξυπνο χαρτί".
103         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
104     """
```

Οι οδηγίες του έξυπνου χαρτιού διαφοροποιούνται ανάλογα με το πλήθος των κινήσεων που έχει πραγματοποιήσει ο παίκτης με τα Χ.

```
105     # πόσες κινήσεις έχει κάνει ο Χ;
106     nbMoves = board.count("X")
```

Τώρα το πρόγραμμά μας μπορεί να ελέγχει πόσες κινήσεις έχει ήδη κάνει ο παίκτης με τα Χ, και να επιλέγει την επόμενη κίνησή του με βάση τις οδηγίες του έξυπνου χαρτιού.

Στην πρώτη κίνηση, το έξυπνο χαρτί προτρέπει τον παίκτη με τα Χ να επιλέξει οποιαδήποτε γωνία. Το πρόγραμμά μας θα επιλέγει την επάνω αριστερά γωνία.

```
107     if nbMoves == 0:
108         # κίνηση 1η: πάνω αριστερά γωνία (θέση 0)
109         return 0
```

Στη δεύτερη κίνηση, ο παίκτης με τα Χ διαθέτει δύο επιλογές: Αν είναι διαθέσιμη η κάτω δεξιά γωνία (που βρίσκεται διαγωνίως απέναντι από τη γωνία που επέλεξε στην πρώτη του κίνηση), τότε θα

επιλέγει να παίξει εκεί. Σε διαφορετική περίπτωση, το πρόγραμμά μας θα επιλέγει την επάνω δεξιά γωνία.

```
110     elif nbMoves == 1:
111         # κίνηση 2η: κάτω δεξιά γωνία (θέση 8), αν
112         # είναι διαθέσιμη, αλλιώς πάνω δεξιά (θέση 2)
113         if board[8] == " ":
114             return 8
115         else:
116             return 2
```

Ο κώδικας για τις επόμενες δύο κινήσεις μας είναι γνώριμος από την `randomPosition`. Αρχικά το πρόγραμμα ελέγχει αν ο παίκτης με τα X (δηλαδή το ίδιο το πρόγραμμα) μπορεί να κάνει τρίλιζα.

```
117     elif nbMoves < 4:
118         # κίνηση 3η, 4η: αν ο "X" κάνει τρίλιζα
119         position = checkPartial("X", board)
120         if position is not None:
121             return position
```

Αν ο X δεν μπορεί να κάνει τρίλιζα, ελέγχει αν μπορεί ο παίκτης με τα O, ώστε να τον εμποδίσει.

```
122         # έλεγξε αν ο "O" μπορεί να κάνει τρίλιζα
123         position = checkPartial("O", board)
124         if position is not None:
125             return position
```

Αν τίποτε από τα παραπάνω δεν μπορεί να γίνει, το πρόγραμμα επιλέγει την πρώτη διαθέσιμη γωνία.

```
126         # αλλιώς παίξε σε οποιαδήποτε ελεύθερη γωνία
127         for corner in (0,2,6,8):
128             if board[corner] == " ":
129                 return corner
```

Αν το παιχνίδι έχει φτάσει μέχρι την τελευταία κίνηση, τότε είναι δεδομένο ότι έχει μείνει μόνο μία διαθέσιμη θέση, η οποία θα αποτελεί το μοναδικό στοιχείο της λίστας `possible`. Αυτή είναι και η θέση που επιλέγει αναγκαστικά το πρόγραμμά μας.

```
130     else:
131         # κίνηση 5η: παίξε στο διαθέσιμο τετράγωνο
132         return board.index(" ")
```

Η μέθοδος `index` εφαρμόζεται σε μια λίστα. Δέχεται σαν παράμετρο ένα στοιχείο της λίστας και επιστρέφει τη θέση του σε αυτή. Αν το στοιχείο δεν υπάρχει στη λίστα τότε προκύπτει *σφάλμα*.

Εδώ η `index` χρησιμοποιείται για να αναζητηθεί το κενό `" "` στη λίστα `board`, δηλαδή για να βρεθεί η τελευταία διαθέσιμη θέση.

Απομένει μόνο ν' αντικαταστήσουμε στο κύριο πρόγραμμα την κλήση της `randomPosition` με μια κλήση στην `paperPosition`, έτσι ώστε το πρόγραμμά μας να παίζει ακολουθώντας πλέον τις οδηγίες του έξυπνου χαρτιού.

```
153     if player == computer:
154         # επιλογή θέσης από το πρόγραμμα
155         position = paperPosition(board)
```

oxo/src/oxo.7.py

Τροποποιήσεις – Επεκτάσεις

5.1 *Η συνάρτηση* `readPosition` *ξεκινά εμφανίζοντας στον παίκτη την τρέχουσα κατάσταση του πίνακα του παιχνιδιού:*

```
# εμφάνιση πίνακα τρίλιζας και πιθανών κινήσεων
print3x3(board)
print3x3(range(9))
```

Αντικαταστήστε την τελευταία από τις εντολές, όπως φαίνεται στον κώδικα που ακολουθεί:

```
# εμφάνιση πίνακα τρίλιζας και πιθανών κινήσεων
print3x3(board)
p = [position if board[position] == " " else "."
     for position in range(9)]
print3x3(p)
```

Εκτελέστε το πρόγραμμα και παρατηρήστε τι συμβαίνει. Τί ακριβώς περιέχει η λίστα `p`*;*

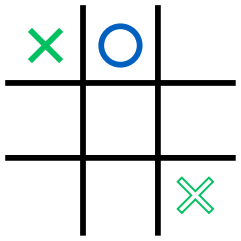
oxo/exercises/oxo.8.py

5.2 *Όταν θελήσαμε να τροποποιήσουμε το πρόγραμμα ώστε να εμφανίζει το νικητή, διαπιστώσαμε ότι στο τέλος του παιχνιδιού η μεταβλητή* `player` *δεν αντιστοιχεί στον παίκτη που κέρδισε, αλλά στον ηττημένο. Αυτό οφείλεται στο γεγονός ότι η τιμή της* `player` *εναλλάσσεται στο τέλος της επανάληψης.*

Μετακινήστε στην αρχή της επανάληψης *τις εντολές που εναλλάσσουν την τιμή της* `player` *και κάντε τις απαραίτητες τροποποιήσεις ώστε το πρόγραμμα να λειτουργεί και πάλι σωστά.*

oxo/exercises/oxo-playermod.py

5.3 *Το έξυπνο χαρτί ξεκινά πάντα παίζοντας με τα X σε γωνιακό τετράγωνο. Αν ο παίκτης με τα O απαντήσει παίζοντας σ' ένα πλευρικό τετράγωνο (θέσεις* 1, 3, 5 *και* 7*, σύμφωνα με την αρίθμησή μας) τότε το έξυπνο χαρτί θα επιλέξει τη γωνία που βρίσκεται διαγωνίως απέναντι από την αρχική (σχήμα 5.6). Αυτή η κίνηση καταλήγει σε ισοπαλία, ενώ υπάρχει καλύτερη που οδηγεί με βεβαιότητα τον παίκτη με τα X στη νίκη.*



Σχήμα 5.6: Η 2η κίνηση του έξυπνου χαρτιού (κάτω δεξιά) αν ο παίκτης με τα O παίξει σε πλευρικό τετράγωνο. Αυτή η κίνηση καταλήγει σε ισοπαλία, ενώ υπάρχει καλύτερη που οδηγεί αυτόματα στη νίκη.

Να επεκτείνετε τις οδηγίες του έξυπνου χαρτιού έτσι ώστε να μην παρουσιάζουν αυτή την "αδυναμία". Όταν ο παίκτης με τα O ξεκινήσει παίζοντας σε πλευρικό τετράγωνο, οι οδηγίες θα πρέπει να επιλέγουν την κατάλληλη κίνηση ώστε ο παίκτης με τα X να κερδίζει. Στη συνέχεια, να επεκτείνετε ανάλογα και τη συνάρτηση `paperPosition`*.*

Αν προτιμάτε να βρείτε έτοιμες τις τροποποιημένες οδηγίες, ώστε να εστιάσετε μόνο στην υλοποίησή τους, μπορείτε να ανατρέξετε στις επεκτάσεις του έξυπνου χαρτιού: `pythonies.mysch.gr/ipaper-ext.pdf`.

oxo/exercises/oxo-more-intelligent.py

5.4 *Όταν φτάσαμε στο σημείο όπου το πρόγραμμά μας έπρεπε να αναλάβει το ρόλο ενός εκ των δύο παικτών, ήταν απαραίτητο να διαθέτουμε έναν μηχανισμό ο οποίος θα επέτρεπε στο πρόγραμμα να ελέγχει αν ένας παίκτης*

έχει τη δυνατότητα να κάνει τρίλιζα. Για τον σκοπό αυτό, αναπτύξαμε τη συνάρτηση `checkPartial`.

Στο βιβλίο του Invent your own computer games with Python, *ο Al Sweigart αφιερώνει επίσης ένα κεφάλαιο στην τρίλιζα και προτείνει τον εξής τρόπο για να ελέγχεται αν ένας παίκτης έχει τη δυνατότητα να κάνει τρίλιζα: για κάθε διαθέσιμη θέση δημιουργείται ένα* αντίγραφο *του πίνακα του παιχνιδιού, συμπληρώνεται η θέση με το σύμβολο του παίκτη και ελέγχεται αν έχει γίνει τρίλιζα. Ουσιαστικά, το πρόγραμμα* δοκιμάζει *τι θα συμβεί στην επόμενη κίνηση.*

Να αναπτύξετε μια νέα υλοποίηση της `checkPartial` *που να χρησιμοποιεί τη μέθοδο που περιγράψαμε για να ελέγξει αν ένας παίκτης έχει τη δυνατότητα να κάνει τρίλιζα. Για να ελέγχει η συνάρτησή σας αν μια κίνηση έχει οδηγήσει σε τρίλιζα, θα πρέπει να καλεί τη συνάρτηση* `check`.

`oxo/exercises/oxo-lookahead.py`

Για να δημιουργήσουμε ένα αντίγραφο μιας λίστας, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `copy()`, η οποία εφαρμόζεται σε μια λίστα κι επιστρέφει ένα αντίγραφό της.

5.5 *Μια αρίθμηση των τετραγώνων της τρίλιζας που πιθανώς να βόλευε περισσότερο τους παίκτες είναι αυτή που φαίνεται στο σχήμα 5.7. Αυτή η αρίθμηση αντιστοιχεί στη διάταξη που έχουν τα αριθμητικά πλήκτρα σε ένα τηλέφωνο ή στο πληκτρολόγιο.*

Τροποποιήστε το πρόγραμμα που αναπτύχθηκε έτσι ώστε ο παίκτης να επιλέγει το τετράγωνο στο οποίο θα παίζει με βάση αυτή την αρίθμηση.

Μια πιθανή προσέγγιση είναι να τροποποιηθεί και η εσωτερική αναπαράσταση, ώστε να υπάρχει μια φυσικότερη αντιστοιχία με την αρίθμηση που αντιλαμβάνεται ο παίκτης.

`oxo/exercises/oxo-dial-representation.py`

Εναλλακτικά, μπορεί η εσωτερική αναπαράσταση να παραμείνει ως έχει και να προστεθεί στον κώδικα ένα επίπεδο "αντιστοίχισης" ανάμεσα στην εσωτερική αναπαράσταση και την αρίθμηση από την σκοπιά του παίκτη.

`oxo/exercises/oxo-dial-mapping.py`

7	8	9
4	5	6
1	2	3

Σχήμα 5.7: Μια διαφορετική αρίθμηση των τετραγώνων της τρίλιζας, με βάση την οποία επιλέγει ο παίκτης το τετράγωνο στο οποίο θα παίξει. Αυτή η αρίθμηση αντιστοιχεί στο κλασικό αριθμητικό πληκτρολόγιο.

5.6 *Το έξυπνο χαρτί παρέχει οδηγίες μόνο για λογαριασμό του παίκτη που παίζει πρώτος. Προσπαθήστε να γράψετε αντίστοιχες οδηγίες για τον παίκτη που παίζει δεύτερος και να τις υλοποιήσετε σε μια συνάρτηση αντίστοιχη της* `paperPosition`.

Αν προτιμάτε να βρείτε έτοιμες τις οδηγίες για τον δεύτερο παίκτη, ώστε να εστιάσετε μόνο στην υλοποίησή τους, μπορείτε να ανατρέξετε στις επεκτάσεις του έξυπνου χαρτιού: `pythonies.mysch.gr/ipaper-ext.pdf`.

Για να χρησιμοποιήσετε τη συνάρτησή σας, θα χρειαστεί να κάνετε τις απαραίτητες τροποποιήσεις ώστε το πρόγραμμα να μπορεί να αναλάβει το ρόλο οποιουδήποτε από τους δύο παίκτες.

`oxo/exercises/oxo-twoplayer.py`

Γιώργος Μπουκέας, Βασίλης Βασιλάκης (2015-2016)

Το παρόν υλικό διατίθεται με άδεια Creative Commons BY-SA 4.0.

`creativecommons.org/licenses/by-sa/4.0/deed.el`