Διαπλανητικό Κουίζ

5

17 Απριλίου 2016

22:55

Στο κεφάλαιο αυτό θα υλοποιήσουμε ένα παιχνίδι ερωτήσεων με θέμα τους πλανήτες του ηλιακού μας συστήματος. Ο παίκτης θα καλείται να βρει έναν πλανήτη από τη θέση του στο ηλιακό σύστημα, να τον μαντέψει από τους γειτονικούς του πλανήτες ή να βάλει τους πλανήτες σε σωστή σειρά, Οι ερωτήσεις που θα τίθενται στον παίκτη δεν θα είναι οι ίδιες κάθε φορά, αλλά θα δημιουργούνται δυναμικά, κατά την εκτέλεση του προγράμματος. Μέσα από το πρόγραμμά μας θα γνωρίσουμε μια πολύ σημαντική έννοια στον προγραμματισμό, τη δομή δεδομένων και πιο συγκεκριμένα μια δομή δεδομένων που ονομάζεται λίστα. Θα έχουμε επίσης την ευκαιρία να εξοικειωθούμε ακόμη περισσότερο με τα υποπρογράμματα, μιας και όλο το πρόγραμμα υλοποιείται με τη βοήθειά τους.

Έννοιες: λίστες, υποπρογράμματα

Όλοι οι Καλοί Χωράνε

Πού θα καταχωρίσω τα ονόματα όλων των πλανητών του ηλιακού συστήματος;

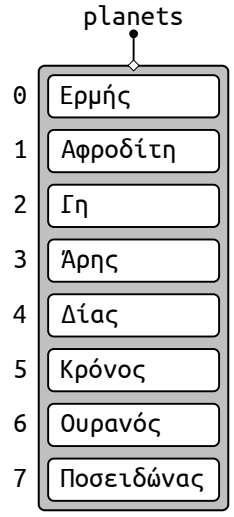
Για να μπορεί το πρόγραμμά μας να "δημιουργεί" ερωτήσεις προς τον χρήστη θα πρέπει να έχει διαθέσιμα τα ονόματα όλων των πλανητών. Μέχρι στιγμής χρειάστηκε να αποθηκεύσουμε τιμές και να αναφερθούμε σε αυτές χρησιμοποιώντας μεταβλητές. Πολύ συχνά όμως χρειάζεται να αποθηκεύσουμε μια ομάδα τιμών με τη μορφή μιας *συλλογής*. Με τον τρόπο αυτό θα έχουμε πρόσβαση σε όλα τα στοιχεία της ομάδας χρησιμοποιώντας ένα κοινό όνομα. Ένα αντικείμενο που μας δίνει αυτή τη δυνατότητα είναι η *λίστα*.

Ας κατασκευάσουμε λοιπόν μια λίστα με τα ονόματα των πλανητών, την οποία θα ονομάσουμε `planets`.

```
1 # η λίστα με τα ονόματα όλων των πλανητών
2 planets = ["Ερμής", "Αφροδίτη", "Γη", "Άρης", "Δίας",
3            "Κρόνος", "Ουρανός", "Ποσειδώνας"]
```

src/planets.1.py

Τα στοιχεία της λίστας καταχωρούνται ακολουθιακά, που σημαίνει ότι παίζει ρόλο η σειρά με την οποία τα εισάγουμε.



Σχήμα 5.1: Η λίστα με τα ονόματα των πλανητών. Οι θέσεις της λίστας είναι αριθμημένες. Με βάση τους αριθμούς των θέσεων μπορούμε ν' αναφερθούμε σε κάθε στοιχείο ξεχωριστά. Προσοχή όμως, η αρίθμηση *ξεκινά από το 0*.

Η Διαμάχη

Με τον Πλούτωνα τί γίνεται; Μερικοί από εμάς μεγάλωσαν μαθαίνοντας ότι ο Πλούτωνας είναι πλανήτης!

Ο Πλούτωνας ανακαλύφθηκε το 1930 και μέχρι το 2006 θεωρούνταν ο ένατος πλανήτης του ηλιακού μας συστήματος. Αν και η Διεθνής Αστρονομική Ένωση τον έχει πια υποβιβάσει σε πλανήτη-νάνο, η παράλειψή του από τη λίστα των πλανητών ίσως δημιουργεί σε κάποιους μια συναισθηματική φόρτιση. Το πρόγραμμά μας λοιπόν, εκδηλώνοντας ευαισθησία, θα ρωτά τον χρήστη αν θέλει ο Πλούτωνας να θεωρηθεί ένας από τους πλανήτες και ανάλογα θα προσθέτει την τιμή `"Πλούτωνας"` στη λίστα με τα ονόματα των πλανητών.

Αρχικά, θα υλοποιήσουμε μια συνάρτηση που θα ρωτά τον παίκτη αν επιθυμεί ο Πλούτωνας να συμπεριληφθεί στους πλανήτες και θα επιστρέφει την τιμή `True` ή `False` ανάλογα με την απάντηση του.

```
def addPluto():
    ''' Ρωτά αν ο Πλούτωνας θα θεωρηθεί πλανήτης
    κι ανάλογα επιστρέφει την τιμή True ή False
    '''
    # ερώτηση στον παίκτη για τον Πλούτωνα
    print("Να θεωρήσουμε τον Πλούτωνα πλανήτη (ν/ο);")
    answer = input()
    # επιστροφή τιμής ανάλογα με την απάντηση
    return answer == "Ν" or answer == "ν"
```

Η τελευταία γραμμή της συνάρτησης επιστρέφει την τιμή της συνθήκης. Είναι ένας αμεσότερος τρόπος να κάνουμε το εξής:

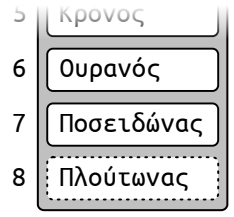
```
    # επιστροφή τιμής ανάλογα με την απάντηση
    if answer == "Ν" or answer == "ν":
        return True
    else:
        return False
```

Το κύριο πρόγραμμα θα καλεί τη συνάρτηση `addPluto()` και θα προσαρτά τον Πλούτωνα στη λίστα, εφόσον η απάντηση του παίκτη είναι θετική.

```
# ερώτηση για προσθήκη του Πλούτωνα
if addPluto():
    # προσάρτηση του Πλούτωνα στο τέλος της λίστας
    planets.append("Πλούτωνας")
```

src/planets.2.py

Παρατηρούμε ότι μπορούμε να προσθέτουμε (ή και να αφαιρούμε) στοιχεία κατά βούληση, τροποποιώντας το μέγεθος της λίστας, δηλαδή το πλήθος των στοιχείων της.



Σχήμα 5.2: Η λίστα των πλανητών, μετά από την προσθήκη ενός ακόμα στοιχείου στο τέλος της.

Η μέθοδος `append()` εφαρμόζεται σε μια λίστα και προσθέτει ένα νέο στοιχείο στο τέλος της. Η αντίστροφη μέθοδος είναι η `pop()`, η οποία εφαρμόζεται σε μια λίστα κι επιστρέφει το τελευταίο στοιχείο, διαγράφοντάς το από τη λίστα.

Πες Μου Πως Σε Λένε Να Σου Πω Που Είσαι

Το παιχνίδι πρέπει να ξεκινά με ένα απλό ερώτημα. Να βρίσκει, για παράδειγμα, ο παίκτης τη θέση ενός πλανήτη στο ηλιακό σύστημα.

Οι ερωτήσεις προς το παίκτη δεν θα είναι πάντα ακριβώς οι ίδιες. Η δημιουργία των ερωτήσεων θα περιλαμβάνει ένα στοιχείο τυχαιότητας, συνήθως την επιλογή ενός από τους πλανήτες. Για τον σκοπό αυτό, θα πρέπει να εισάγουμε την κατάλληλη βιβλιοθήκη.

```
1  import random
```

Θα υλοποιούμε κάθε ερώτημα με χρήση συναρτήσεων. Έτσι θα διατηρούμε τον κώδικα μας ξεκάθαρο, αλλά και θα ξαναχρησιμοποιούμε, όπου είναι δυνατόν, τα ίδια υποπρογράμματα.

Για το πρώτο ερώτημα θα υλοποιήσουμε μια συνάρτηση η οποία δέχεται ως παράμετρο τη λίστα των πλανητών, επιλέγει από αυτή τυχαία έναν πλανήτη και, αφού εμφανίσει το όνομά του στο χρήστη, του ζητά να καθορίσει τη θέση του στο ηλιακό σύστημα.

Ένας τρόπος να επιλεχθεί τυχαία ένας πλανήτης είναι να επιλεχθεί πρώτα ένας τυχαίος ακέραιος που θα αντιστοιχεί στη *θέση* του πλανήτη μέσα στη λίστα. Η θέση αυτή θα χρησιμοποιηθεί για να προσδιορίστει και το όνομα του πλανήτη.

```
11 def findByName(planets):
12     ''' Εμφανίζει το όνομα ενός πλανήτη και ζητά
13     από τον παίκτη τη θέση του στο ηλιακό σύστημα.
14     planets: λίστα με τα ονόματα όλων των πλανητών
15     '''
16     # επιλογή θέσης τυχαίου πλανήτη
17     nbPlanets = len(planets)
18     position = random.randint(0, nbPlanets - 1)
```

Η συνάρτηση `len()` επιστρέφει το πλήθος των στοιχείων της λίστας.

Η επιλογή της τυχαίας θέσης χρειάζεται προσοχή. Οι πιθανές τιμές για την μεταβλητή `position` κυμαίνονται ανάμεσα στο μηδέν και το πλήθος των στοιχείων της λίστας *μειωμένο κατά ένα*. Αυτό συμβαίνει γιατί η αρίθμηση των θέσεων σε μια λίστα *ξεκινά από το μηδέν*.

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 είναι το `planets[0]` και αντιστοιχεί στην τιμή `"Ερμής"`, το δεύτερο είναι το `planets[1]` και αντιστοιχεί στην τιμή `"Αφροδίτη"`, κ.ο.κ.

Τώρα γνωρίζουμε ότι ο πλανήτης που έχουμε επιλέξει βρίσκεται στη θέση `position`. Το όνομα του πλανήτη βρίσκεται στην αντίστοιχη θέση μέσα στη λίστα `planets`.

```
19     # αναφορά στο όνομα του πλανήτη
20     # μέσω της θέσης του στη λίστα
21     planet = planets[position]
```

Οι θέσεις μιας λίστας αριθμούνται και αντίστροφα, ξεκινώντας από το τέλος, οπότε π.χ. στο τελευταίο στοιχείο μπορούμε να αναφερθούμε και ως `planets[-1]`.

Στη συνέχεια η συνάρτηση θα εμφανίζει την ερώτηση στον παίκτη και θα ζητάει την απάντησή του.

```
22     # ερώτηση στον παίκτη
23     print("Σε ποια σειρά βρίσκεται ο πλανήτης ",
24           planet, " σε σχέση με τον Ήλιο;", sep = "")
25     answer = int(input())
```

Για να αναφερθούμε σ' ένα στοιχείο της λίστας μπορούμε να χρησιμοποιήσουμε ανάμεσα στις αγκύλες οποιαδήποτε *ακέραια έκφραση*, όπως εδώ την `position`. Η τιμή της έκφρασης καθορίζει τη θέση του στοιχείου της λίστας στο οποίο αναφερόμαστε.

Η απάντηση θα πρέπει να ελεγχθεί και να εμφανίζεται είτε μήνυμα επιβράβευσης είτε η σωστή απάντηση, σε περίπτωση αποτυχίας.

```
26      # έλεγχος της απάντησης του παίκτη
27      if answer == position + 1:
28          # αν η απάντηση είναι σωστή
29          print("Μπράβο, το βρήκες!")
30      else:
31          # εμφάνιση της σωστής απάντησης
32          print("Ο πλανήτης ", planet, " είναι ο ",
33                position + 1, "ος πλανήτης.", sep = "")
```

Η παράμετρος `sep` της **`print`** καθορίζει τι θα εμφανίζεται ανάμεσα στα επιμέρους μηνύματα της **`print`**. Η προκαθορισμένη τιμή της είναι η `" "`, γι' αυτό κανονικά εμφανίζεται ένα κενό. Αν επιθυμούμε να αλλάξει αυτή η προκαθορισμένη συμπεριφορά τότε αρκεί να επανορίσουμε την `sep`.

Εδώ υπάρχει και πάλι ένα λεπτό σημείο. Η θέση ενός πλανήτη στο ηλιακό σύστημα είναι κάποια τιμή από το 1 μέχρι και το πλήθος των πλανητών. Αντίθετα, η θέση του στη λίστα είναι κάποια τιμή από το 0 μέχρι και το πλήθος των πλανητών μειωμένο κατά 1. Η διαφορά αυτή οφείλεται στο γεγονός ότι οι άνθρωποι ξεκινούν την αρίθμηση θέσεων από το 1, ενώ στις λίστες η αρίθμηση ξεκινά από το 0. Κατά συνέπεια, η θέση ενός πλανήτη στο ηλιακό σύστημα είναι κατά μια μονάδα μεγαλύτερη από τη θέση του στη λίστα και γι' αυτό χρειάζεται να προσθέσουμε μια μονάδα στη μεταβλητή `position` πριν την συγκρίνουμε με την απάντηση `answer` του παίκτη.

Ώρα να ξεκινήσουμε το παιχνίδι, εμφανίζοντας το πρώτο ερώτημα στον παίκτη.

```
41  # εύρεση του πλανήτη από το όνομα
42  print("\nΕρώτηση 1:")
43  findByName(planets)
```

src/planets.3.py

Το `\n` είναι ένας "κωδικός" που αναπαριστά την αλλαγή γραμμής. Εδώ θέλουμε να υπάρχει μια κενή γραμμή ανάμεσα στο μήνυμα που θα εμφανιστεί και τις προηγούμενες γραμμές.

Πες Μου Που Είσαι Να Σου Πω Πως Σε Λένε

Γίνεται ν' αντιστρέψουμε τώρα το ερώτημα; Να δίνεται στο χρήστη η θέση ενός πλανήτη και να ζητείται τ' όνομά του.

Θα υλοποιήσουμε μια ακόμα συνάρτηση, παρόμοια με την προηγούμενη, η οποία επιλέγει τυχαία έναν πλανήτη και, αφού εμφανίσει το ερώτημα στον παίκτη, ζητά και ελέγχει την απάντησή του.

```
34  def findByPosition(planets):
35      ''' Εμφανίζει τη θέση ενός πλανήτη στο ηλιακό
36      σύστημα και ζητά από τον παίκτη το όνομά του.
37      planets: λίστα με τα ονόματα όλων των πλανητών
38      '''
39      # επιλογή τυχαίου πλανήτη
40      nbPlanets = len(planets)
41      position = random.randint(0, nbPlanets - 1)
42      planet = planets[position]
```

```
43      # ερώτηση στον παίκτη
44      print("Ποιος είναι ο ", position + 1,
45            "ος πλανήτης μετά τον Ήλιο;", sep = "")
46      answer = input()
47      # έλεγχος της απάντησης του παίκτη
48      if answer == planet:
49          # αν η απάντηση είναι σωστή
50          print("Μπράβο, το βρήκες!")
51      else:
52          # εμφάνιση της σωστής απάντησης
53          print("Είναι ο πλανήτης ", planet, ".", sep = "")
```

src/planets.4.py

Τι θα γίνει όμως σε περίπτωση που ο παίκτης γράψει το όνομα του σωστού πλανήτη με κεφαλαία γράμματα ή αφήσει κενά στην απάντησή του; Παρόλο που ο παίκτης θα έχει ουσιαστικά απαντήσει σωστά, το πρόγραμμά μας θα του εμφανίζει το αντίθετο, αφού για να θεωρηθεί η απάντηση σωστή θα πρέπει να είναι πανομοιότυπη με το όνομα του πλανήτη, όπως εμφανίζεται στη λίστα των πλανητών.

Ένα μικρό και πολύ συνηθισμένο τρικ για να παρακάμψουμε αυτό το πρόβλημα είναι να συγκρίνουμε την απάντηση του παίκτη και το όνομα του σωστού πλανήτη αφού πρώτα τα μετατρέψουμε σε πεζά γράμματα, διαγράφοντας μάλιστα και τυχόν κενά που υπάρχουν στην αρχή ή στο τέλος της απάντησης.

```
    if answer.lower().strip() == name.lower():
```

Η μέθοδος `lower()` εφαρμόζεται σε μια αλφαριθμητική τιμή και μετατρέπει τους χαρακτήρες της σε πεζούς. Η αντίστροφη μέθοδος είναι η `upper()`.

Η μέθοδος `strip()` εφαρμόζεται σε μια αλφαριθμητική τιμή και αφαιρεί τα κενά που πιθανώς να βρίσκονται στην αρχή και το τέλος της.

Μετατρέποντας τις δύο τιμές σε μια κοινή αναπαράσταση πριν τις συγκρίνουμε μετριάζουμε σε σημαντικό βαθμό το πρόβλημα. Έχετε όμως υπόψη ότι δεν είναι και τόσο απλό να λυθεί εντελώς. Για παράδειγμα, αν ο παίκτης κάνει κάποιο ορθογραφικό λάθος ή δεν τονίσει την απάντησή του τότε και πάλι δεν θα θεωρηθεί σωστή.

Στο κύριο πρόγραμμα, μπορούμε να προσθέσουμε τη νέα ερώτηση στο παιχνίδι μας.

```
64  # εύρεση του πλανήτη από τη θέση
65  print("\nΕρώτηση 2:")
66  findByPosition(planets)
```

src/planets.4.py

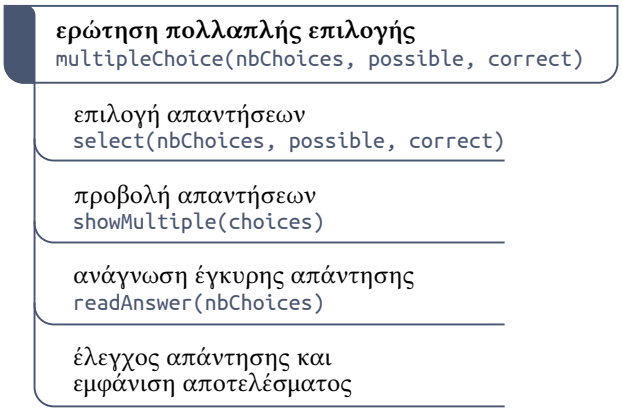
Η Δυσκολία Στη Ζωή Είναι Να Διαλέξεις

Όταν ο παίκτης πληκτρολογεί τις απαντήσεις του μπαίνουμε σε μπελάδες. Ας κάνουμε τις ερωτήσεις πολλαπλής επιλογής.

Μια ερώτηση πολλαπλής επιλογής συνοδεύεται από ένα πλήθος προκαθορισμένων απαντήσεων, από τις οποίες ο χρήστης επιλέγει εκείνη που θεωρεί σωστή. Για να παράγουμε μια ερώτηση πολλαπλής επιλογής, θα πρέπει αρχικά να γνωρίζουμε ποιες είναι οι πιθανές απαντήσεις, πόσες από αυτές θα προσφερθούν ως επιλογές στο χρήστη και, φυσικά, ποια είναι η σωστή απάντηση.

Θα μπορούσαμε λοιπόν απευθείας να υλοποιήσουμε μια συνάρτηση η οποία δέχεται τις παραπάνω παραμέτρους και παράγει μια ερώτηση πολλαπλής επιλογής.

Ωστόσο, η παραγωγή μιας ερώτησης πολλαπλής επιλογής περιλαμβάνει συγκεκριμένα στάδια, μερικά από τα οποία δεν είναι τετριμμένα και μπορούν να υλοποιηθούν με αρκετούς διαφορετικούς τρόπους. Για τον σκοπό αυτό, κάθε στάδιο (εκτός από το τελευταίο που είναι απλό) θα υλοποιηθεί σε ξεχωριστό υποπρόγραμμα.



Σχήμα 5.3: Το πρόβλημα της παραγωγής μιας ερώτησης πολλαπλής επιλογής αναλύεται σε μικρότερα προβλήματα. Στην αναπαράσταση αυτή φαίνονται και τα υποπρογράμματα που θα δημιουργηθούν στη συνέχεια για τα προβλήματα αυτά.

Για την επιλογή των πιθανών απαντήσεων θα υλοποιήσουμε μια συνάρτηση που δέχεται τη λίστα με όλες τις πιθανές απαντήσεις, καθώς και την σωστή απάντηση, επιλέγει τυχαία ένα ορισμένο πλήθος από αυτές και τις επιστρέφει σε μια λίστα, ώστε να μπορούν στη συνέχεια να εμφανιστούν στο χρήστη.

```
34 def select(nbChoices, possible, correct):
35     ''' Επιλέγει από τη λίστα πιθανών απαντήσεων
36     ένα ορισμένο πλήθος και τις επιστρέφει σε λίστα.
37     nbChoices: πλήθος απαντήσεων που θα επιλεχθούν
38     possible: λίστα όλων των πιθανών απαντήσεων,
39         πρέπει len(possible) >= nbChoices
40     correct: η σωστή απάντηση
41     '''
42     # διάλεξε τυχαία απαντήσεις
43     answers = random.sample(possible, nbChoices)
```

Η συνάρτηση `sample()` της βιβλιοθήκης `random` δέχεται σαν παράμετρο μια λίστα και το πλήθος των τιμών που θα επιλεχθούν από αυτή και επιστρέφει μια νέα λίστα που περιέχει μια τυχαία επιλογή στοιχείων της πρώτης. Αν θέλουμε να επιλέξουμε τυχαία μόνο ένα στοιχείο μιας λίστας, χρησιμοποιούμε την συνάρτηση `choice()`, της ίδιας βιβλιοθήκης.

Επειδή κάνουμε τα πρώτα μας βήματα στη διαχείριστη λιστών, χρησιμοποιήσαμε έναν "έτοιμο" τρόπο να επιλέξουμε ένα υποσύνολο των πιθανών απαντήσεων. Υπάρχουν και αρκετοί εναλλακτικοί τρόποι τους οποίους θα μπορέσετε να δοκιμάσετε στις ασκήσεις.

Η συνάρτηση δεν έχει ακόμα ολοκληρωθεί. Στη λίστα `answers` των απαντήσεων που θα επιλεχθούν θα πρέπει να περιέχεται οπωσδήποτε και η σωστή απάντηση, κάτι το οποίο δεν έχουμε εξασφαλίσει.

Σε περίπτωση που η σωστή απάντηση δεν περιέχεται στην `answers`, θα πρέπει να εισαχθεί σε μια *τυχαία* θέση της λίστας, *αντικαθιστώντας* το στοιχείο που ήδη βρίσκεται εκεί.

```
    # αν η σωστή απάντηση δεν υπάρχει στη λίστα
    if correct not in answers:
        # διάλεξε τυχαία θέση για την σωστή απάντηση
        index = random.randint(0, nbChoices - 1)
        # εισαγωγή της σωστής απάντησης στη λίστα
        answers[index] = correct
    # επιστροφή λίστας απαντήσεων
    return answers
```

Ο τελεστής `in` ελέγχει αν μια τιμή υπάρχει σε μια λίστα και επιστρέφει αντίστοιχα την τιμή `True` ή `False`.

Χρησιμοποιώντας μια εντολή όπως η `answers[index] = correct` *τροποποιείται* ένα στοιχείο της λίστας, καθώς μια νέα τιμή αντικαθιστά την προηγούμενη. Είναι ένα βασικό χαρακτηριστικό των λιστών ότι τα στοιχεία τους μπορούν να τροποποιηθούν.

Για την εμφάνιση των εναλλακτικών επιλογών θα υλοποιήσουμε μια συνάρτηση που δέχεται σαν παράμετρο τη λίστα με τις πιθανές απαντήσεις και τις εμφανίζει αριθμημένες στον παίκτη.

```
def showMultiple(answers):
    ''' Εμφανίζει αριθμημένες τις πιθανές απαντήσεις
    μιας ερώτησης, σε στυλ πολλαπλής επιλογής.
    answers: λίστα απαντήσεων
    '''
    number = 1
    # σάρωση λίστας πιθανών απαντήσεων
    for answer in answers:
        print("(", number, ") ", answer,
              sep = "", end = " ")
        number = number + 1
```

Η εντολή `for` είναι μια εντολή επανάληψης που διατρέχει τα στοιχεία μιας ακολουθίας τιμών, όπως μια λίστα ή οι χαρακτήρες μιας αλφαριθμητικής τιμής, 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`.

Όπως και στην εντολή `while` έτσι και στη `for` στοιχίζουμε δεξιότερα τις εντολές που περιέχονται σε αυτήν.

Μια ακόμα συνάρτηση θα ζητάει την απάντηση του παίκτη, θα ελέγχει ότι είναι έγκυρη, δηλαδή εντός του πλήθους των πιθανών επιλογών, και θα την επιστρέφει.

```
def readAnswer(nbChoices):
    ''' Ζητάει την απάντηση του παίκτη,
    μέχρι να είναι έγκυρη, και την επιστρέφει.
    nbChoices: πλήθος έγκυρων απαντήσεων
    '''
    # επανάληψη: μέχρι να δοθεί έγκυρη απάντηση
    while True:
        # ανάγνωση απάντησης
        choice = int(input())
        # έλεγχος εγκυρότητας
        if choice < 1 or choice > nbChoices:
            print("Επίλεξε μια έγκυρη απάντηση",
                  "από 1 μέχρι", nbChoices, sep = "")
        else:
            break
    # η απάντηση επιστρέφεται
    return choice
```

Η τελευταία συνάρτηση που θα υλοποιήσουμε θα καλεί όλες τις προηγούμενες και θα ελέγχει αν η απάντηση που έδωσε ο παίκτης είναι σωστή ή λανθασμένη.

```
80  def multipleChoice(nbChoices, possible, correct):
81      ''' Ερώτηση πολλαπλής επιλογής: εμφανίζει τις
82      πιθανές απαντήσεις, ζητάει την επιλογή του παίκτη
83      και ελέγχει αν είναι σωστή ή λανθασμένη.
84      nbChoices: πλήθος απαντήσεων που θα εμφανιστούν
85      possible: λίστα όλων των πιθανών απαντήσεων
86      correct: η σωστή απάντηση
87      '''
88      # επιλογή απαντήσεων
89      answers = select(nbChoices, possible, correct)
90      # προβολή απαντήσεων
91      showMultiple(answers)
92      # ανάγνωση έγκυρης απάντησης
93      playerChoice = readAnswer(nbChoices)
94      # έλεγχος και εμφάνιση αποτελέσματος
95      if answers[playerChoice - 1] == correct:
96          print("Μπράβο, το βρήκες!")
97      else:
98          print("Η σωστή απάντηση είναι: ",
99                correct, ".", sep = "")
```

Μια σημαντική παρατήρηση είναι ότι όλα τα προηγούμενα υποπρογράμματα που υλοποιήσαμε είναι τελείως ανεξάρτητα από το πρόγραμμα με τους πλανήτες. Μπορούμε να τα χρησιμοποιήσουμε για να εμφανίζουμε σε ένα πρόγραμμα ερωτήσεις τύπου πολλαπλής επιλογής ανεξάρτητα από το περιεχόμενό τους.

Πλέον μπορούμε να τροποποιήσουμε την συνάρτηση που αντιστοιχεί στο δεύτερο ερώτημα. Το αρχικό τμήμα που επιλέγει έναν τυχαίο πλανήτη και θέτει το ερώτημα στον παίκτη είναι ακριβώς το ίδιο, αλλά ακολουθείται από την κλήση της συνάρτησης για τη δημιουργία ερώτησης πολλαπλής επιλογής.

```
100 def findByPosition(planets):
101     ''' Εμφανίζει τη θέση ενός πλανήτη στο ηλιακό
102     σύστημα και ζητά από τον παίκτη το όνομά του.
103     planets: λίστα με τα ονόματα όλων των πλανητών
104     '''
105     # επιλογή τυχαίου πλανήτη
106     nbPlanets = len(planets)
107     position = random.randint(0, nbPlanets - 1)
108     planet = planets[position]
109     # ερώτηση πολλαπλής επιλογής στον παίκτη
110     print("Ποιος είναι ο ", position + 1,
111           "ος πλανήτης μετά τον Ήλιο;", sep = "")
112     multipleChoice(4, planets, planet)
```

src/planets.5.py

Παρατηρήστε τις παραμέτρους με τις οποίες καλούμε την συνάρτηση `multipleChoice()`. Καθορίζουν ότι θα εμφανιστούν στο χρήστη 4 επιλογές, ενώ η λίστα των πιθανών απαντήσεων περιλαμβάνει όλους τους πλανήτες. Ως σωστή απάντηση προσδιορίζεται βέβαια το όνομα του πλανήτη που έχει επιλεχθεί.

Πες Μου Το Γείτονά Σου Να Σου Πω Ποιος Είσαι

Θα μπορούσαμε να ζητάμε από τον παίκτη να προσδιορίσει έναν πλανήτη από τους γειτονικούς του πλανήτες;

Για το ερώτημα αυτό θα υλοποιήσουμε μια συνάρτηση η οποία δέχεται σαν παράμετρο τη λίστα των πλανητών, εμφανίζει τους γείτονες ενός πλανήτη και ζητά από τον παίκτη να προσδιορίσει για ποιον πλανήτη πρόκειται. Θα ξεκινήσουμε τη συνάρτηση επιλέγοντας όπως πάντα έναν τυχαίο πλανήτη.

```
113 def findByNeighbours(planets):
114     ''' Ζητά από τον παίκτη να βρει έναν πλανήτη,
115     εμφανίζοντας τους γειτονικούς του πλανήτες.
116     planets: λίστα με τα ονόματα όλων των πλανητών
117     '''
118     # επιλογή τυχαίου πλανήτη
119     nbPlanets = len(planets)
120     position = random.randint(0, nbPlanets - 1)
121     planet = planets[position]
```

Ο προηγούμενος πλανήτης βρίσκεται στη θέση `position - 1` και ο επόμενος στη θέση `position + 1`. Τί γίνεται όμως αν έχει επιλεχθεί ο πρώτος πλανήτης στη θέση 0; Τότε δεν έχει νόημα η αναφορά στη θέση `position - 1`. Ομοίως, αν ο πλανήτης βρίσκεται στην τελευταία θέση της λίστας τότε δεν έχει νόημα η αναφορά στη θέση `position + 1`. Το πρόγραμμά μας θα πρέπει να εξετάζει το ενδεχόμενο ο πλανήτης να βρίσκεται στα άκρα του ηλιακού συστήματος και να προσαρμόζει κατάλληλα το ερώτημα προς τον παίκτη.

```
122     # ερώτημα προς τον παίκτη
123     print("Ποιος πλανήτης", end = " ")
124     # έλεγχος για την ύπαρξη γειτόνων
125     if position == 0:
126         # υπάρχει μόνο επόμενος πλανήτης
127         next = planets[position + 1]
128         print("βρίσκεται πριν από τον πλανήτη ",
129               next, ";", sep = "")
130     elif position == nbPlanets - 1:
131         # υπάρχει μόνο προηγούμενος πλανήτης
132         previous = planets[position - 1]
133         print("βρίσκεται μετά από τον πλανήτη ",
134               previous, ";", sep = "")
```

```
135     else:
136         # υπάρχουν και οι δύο γειτονικοί πλανήτες
137         next = planets[position + 1]
138         previous = planets[position - 1]
139         print("βρίσκεται ανάμεσα στους πλανήτες ",
140               previous, " και ", next, ";", sep = "")
```

Και αυτό το ερώτημα θα είναι πολλαπλής επιλογής, επομένως θα γίνει χρήση των υποπρογραμμάτων που υλοποιήσαμε προηγουμένως. Θα εμφανιστούν 4 επιλογές από τη λίστα των πλανητών, συμπεριλαμβανομένης και της σωστής.

```
141     # πολλαπλή επιλογή
142     multipleChoice(4, planets, planet)
```

Πλέον μπορούμε να εμφανίσουμε το ερώτημα στον παίκτη.

```
156 # εύρεση του πλανήτη από τους γείτονές του
157 print("\nΕρώτηση 3:")
158 findByNeighbours(planets)
```

src/planets.6.py

Παίζοντας, θα διαπιστώσουμε ότι το πρόγραμμά μας δεν λειτουργεί ακριβώς όπως θα έπρεπε. Μερικές φορές οι γειτονικοί πλανήτες εμφανίζονται τόσο στην ερώτηση όσο και στις πιθανές απαντήσεις. Θα πρέπει να διορθώσουμε τη συνάρτηση έτσι ώστε η λίστα πιθανών απαντήσεων να μην είναι ολόκληρη η λίστα των πλανητών, όπως είναι τώρα, αλλά ένα αντίγραφό της από το οποίο θα έχουμε αφαιρέσει τους γείτονες του ζητούμενου πλανήτη.

Ας ξεκινήσουμε κατασκευάζοντας το αντίγραφο, το οποίο είναι απαραίτητο προκειμένου να μην αλλοιωθεί η αρχική λίστα πλανητών.

```
122     # πιθανές απαντήσεις: αντίγραφο λίστας πλανητών
123     answers = planets.copy()
```

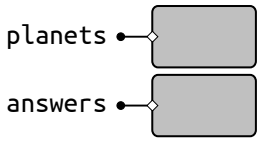
Τώρα, στα σημεία όπου προσδιορίζουμε τους γειτονικούς πλανήτες θα πρέπει να φροντίζουμε ώστε αυτοί να αφαιρούνται από τη λίστα πιθανών απαντήσεων.

```
128         # υπάρχει μόνο επόμενος πλανήτης
129         next = planets[position + 1]
130         answers.remove(next)
```

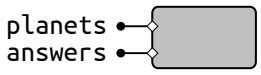
```
134         # υπάρχει μόνο προηγούμενος πλανήτης
135         previous = planets[position - 1]
136         answers.remove(previous)
```

```
140         # υπάρχουν και οι δύο γειτονικοί πλανήτες
141         next = planets[position + 1]
142         previous = planets[position - 1]
143         answers.remove(next)
144         answers.remove(previous)
```

Η μέθοδος `copy()` εφαρμόζεται σε μια λίστα κι επιστρέφει ένα αντίγραφό της. Εδώ η δημιουργία αντιγράφου είναι απαραίτητη γιατί δεν θέλουμε να τροποποιηθεί η αρχική λίστα `planets`.



Σχήμα 5.4: Η δημιουργία αντιγράφου με την `answers = planets.copy()` εξασφαλίζει ότι οποιαδήποτε τροποποίηση της `answers` αφήνει την `planets` αμετάβλητη, αφού πρόκειται για διαφορετικές λίστες.



Σχήμα 5.5: Αντίθετα, με την εντολή `answers = planets` θα δίναμε απλά δύο ονόματα στην ίδια λίστα. Οποιαδήποτε αλλαγή στη λίστα αντικατοπτρίζεται και στα δύο ονόματα.

Η μέθοδος `remove()` εφαρμόζεται σε μια λίστα. Δέχεται σαν παράμετρο ένα στοιχείο της λίστας και το αφαιρεί απ' αυτή. Αν το στοιχείο δεν υπάρχει στη λίστα τότε προκύπτει σφάλμα.

Οι πιθανές απαντήσεις στην ερώτηση πολλαπλής επιλογής θα πρέπει πλέον να προέρχονται από τη νέα λίστα `answers`.

```
147     # πολλαπλή επιλογή
148     multipleChoice(4, answers, planet)
```

src/planets.7.py

Here Comes The Sun

Κάτι άλλο που θα μπορούσαμε να ζητάμε από τον παίκτη είναι να επιλέξει ποιος από τους πλανήτες που του δίνονται είναι πιο κοντά στον Ήλιο.

Η συνάρτηση που θα υλοποιήσουμε για το σκοπό αυτό θα ξεκινά επιλέγοντας τυχαία τον πλανήτη που θα αποτελεί την σωστή απάντηση, δηλαδή εκείνον που θα βρίσκεται πιο κοντά στον Ήλιο. Για να υπάρχουν τουλάχιστον τρεις ακόμα πλανήτες μετά τον αρχικό, η επιλογή του δεν θα γίνεται σε όλο το εύρος των στοιχείων της λίστας, αλλά σε ένα πιο περιορισμένο διάστημα.

```
149 def closestSun(planets):
150     ''' Εμφανίζει τα ονόματα 4 πλανητών και καλεί τον
151     παίκτη να επιλέξει τον κοντινότερο στον Ήλιο.
152     planets: λίστα με τα ονόματα όλων των πλανητών
153     '''
154     # επιλογή τυχαίου πλανήτη
155     # εξαιρούνται οι τρεις τελευταίοι πλανήτες
156     nbPlanets = len(planets)
157     position = random.randint(0, nbPlanets - 4)
158     planet = planets[position]
```

Για να συμπληρωθεί η ερώτηση πρέπει να δημιουργήσουμε τη λίστα των πιθανών απαντήσεων. Σε αντίθεση με προηγούμενα ερωτήματα, αυτή δεν θα προκύψει από ολόκληρη τη λίστα με τους πλανήτες, αλλά μόνο από τον επιλεγμένο πλανήτη κι αυτούς που ακολουθούν.

```
159     # νέα λίστα με τις πιθανές απαντήσεις:
160     # είναι οι πλανήτες από τον επιλεγμένο και μετά
161     following = planets[position : ]
```

Εναλλακτικά, η λίστα `following` μπορεί να προκύψει ως εξής:

```
    following = planets[position : len(planets)]
```

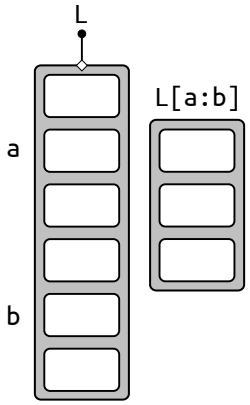
Ας εμφανίσουμε την ερώτηση στον παίκτη.

```
162     # ερώτηση πολλαπλής επιλογής στον παίκτη
163     print("Ποιος πλανήτης είναι πιο κοντά στον Ήλιο;")
164     multipleChoice(4, following, planet)
```

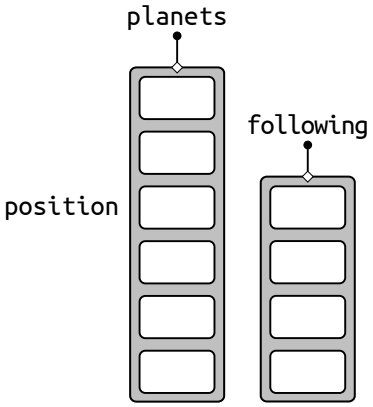
Ώρα για λίγο παιχνίδι ακόμα.

```
181 # εύρεση του κοντινότερου στον ήλιο
182 print("\nΕρώτηση 4:")
183 closestSun(planets)
```

src/planets.8.py



Σχήμα 5.6: Το τμήμα `L[a:b]` είναι μια νέα λίστα που προκύπτει από τον *τεμαχισμό* της λίστας `L`. Ο τεμαχισμός σταματά *πριν* το στοιχείο στη θέση `b`. Αν παραλειφθεί η θέση εκκίνησης τότε ο τεμαχισμός ξεκινά από την αρχή της λίστας. Αν παραλειφθεί η θέση τερματισμού τότε ο τεμαχισμός σταματά στο τέλος της λίστας, περιλαμβάνοντας το τελευταίο στοιχείο.



Σχήμα 5.7: Η λίστα `following` προκύπτει από τον τεμαχισμό της λίστας των πλανητών με την έκφραση `planets[position : ]`.

Ο Παράταιρος

Ας ρωτήσουμε κάτι πιο περίπλοκο. Ας καλέσουμε τον παίκτη να ταξιδέψει νοερά ανάμεσα σε δύο πλανήτες και να σκεφτεί ποιους πλανήτες θα συναντήσει στη διαδρομή. Εμείς θα τον καλούμε να βρει έναν πλανήτη που δεν θα είναι μέρος της διαδρομής.

Θα υλοποιήσουμε μια ακόμα συνάρτηση, στην οποία καταρχήν θα επιλέγουμε τους δύο πλανήτες που θα αποτελέσουν την αφετηρία και τον τερματισμό του ταξιδιού.

```
165 def between(planets):
166     ''' Εμφανίζει 4 πλανήτες, από τους οποίους μόνο
167     οι 3 αποτελούν τμήμα μιας διαδρομής. Καλεί τον
168     παίκτη να επιλέξει τον 4ο "παράταιρο" πλανήτη.
169     planets: λίστα με τα ονόματα όλων των πλανητών
170     '''
171     # τυχαία επιλογή αφετηρίας και τερματισμού
172     nbPlanets = len(planets)
173     start = random.randint(0, nbPlanets - 5)
174     stop = start + 4
```

Παρατηρήστε ότι η αφετηρία δεν επιλέγεται απ' όλο το εύρος της λίστας, αλλά από ένα περιορισμένο διάστημα, ώστε να υπάρχουν μετά από αυτή τουλάχιστον τέσσερις πλανήτες. Ο τέταρτος από αυτούς είναι ο προορισμός της διαδρομής.

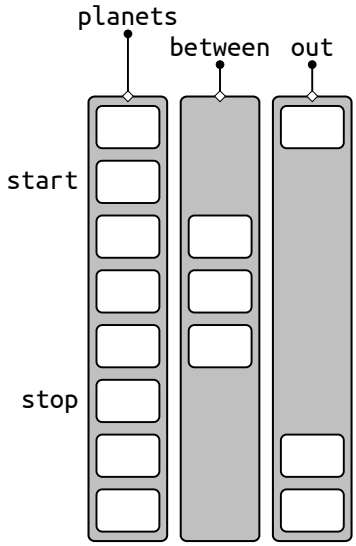
Οι τρεις ενδιάμεσοι πλανήτες τοποθετούνται σε μια λίστα για να χρησιμοποιηθούν ως (λανθασμένες) απαντήσεις στην ερώτηση πολλαπλής επιλογής. Οι πλανήτες που δεν αποτελούν κομμάτι της διαδρομής συγκεντρώνονται επίσης σε μια λίστα, από την οποία θα επιλεχθεί τυχαία μια σωστή απάντηση.

```
175     # πλανήτες μεταξύ αφετηρίας και τερματισμού
176     between = planets[start + 1 : stop]
177     # υπόλοιποι πλανήτες
178     out = planets[ : start] + planets[stop + 1 : ]
179     # τυχαία επιλογή μιας σωστής απάντησης
180     correct = random.choice(out)
```

Πλέον είμαστε έτοιμοι να εμφανίσουμε την ερώτηση στον παίκτη μαζί με τις πιθανές επιλογές.

```
181     # ερώτημα προς τον παίκτη
182     print("Ποιον πλανήτη δεν θα συναντήσουμε ",
183           "ταξιδεύοντας από τον πλανήτη ",
184           planets[start], " στον πλανήτη ",
185           planets[stop], ";", sep = "")
186     # πολλαπλή επιλογή
187     multipleChoice(4, between + [correct], correct)
```

Η λίστα των πιθανών απαντήσεων που δίνεται ως παράμετρος στην συνάρτηση `multipleChoice`, είναι η συνένωση της λίστας `between`



Σχήμα 5.8: Η λίστα `between` των ενδιάμεσων πλανητών και η λίστα `out` των πλανητών που δεν αποτελούν τμήμα της διαδρομής. Και οι δύο προκύπτουν από τεμαχισμούς της `planets` και συνενώσεις.

Ο τελεστής `+` χρησιμοποιείται ανάμεσα σε δύο λίστες. Το αποτέλεσμα της πράξης είναι μια νέα λίστα που περιέχει όλα τα στοιχεία των αρχικών με την ίδια σειρά.

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

Εμείς μέχρι στιγμής δεν έχουμε χρησιμοποιήσει την `choice()` για να επιλέγουμε πλανήτες γιατί στις περισσότερες περιπτώσεις δεν χρειαζόμαστε μόνο έναν τυχαίο πλανήτη αλλά και τη θέση του μέσα στη λίστα. Έτσι, επιλέγουμε πρώτα τυχαία τη θέση ενός πλανήτη και μετά αναφερόμαστε σε αυτόν μέσω της θέσης του.

των ενδιάμεσων πλανητών και μιας λίστας που αποτελείται από ένα μόνο στοιχείο: την σωστή απάντηση `correct`.

Στη συνέχεια θα προσθέσουμε την ερώτηση στο πρόγραμμά μας.

```
207 # εύρεση του "παράταιρου" πλανήτη
208 print("\nΕρώτηση 5:")
209 between(planets)
```

src/planets.9.py

Όλα Σε Τάξη

Ας τελειώσουμε μ' ένα ερώτημα που θα εμφανίζει στον παίκτη ανακατεμένα τα ονόματα τεσσάρων διαδοχικών πλανητών και θα του ζητά να τα βάλει σε σειρά, ξεκινώντας από τον κοντινότερο πλανήτη στον Ήλιο.

Η συνάρτηση που θα υλοποιήσουμε επιλέγει αρχικά τη θέση του πρώτου πλανήτη της τετράδας και δημιουργεί μια νέα λίστα που περιέχει τον πλανήτη αυτόν και τους τρεις που τον διαδέχονται.

```
188 def outOfOrder(planets):
189     ''' Καλεί τον παίκτη να βάλει στη σειρά
190     τα ονόματα 4 διαδοχικών πλανητών.
191     planets: λίστα με τα ονόματα όλων των πλανητών
192     '''
193     # τυχαία επιλογή θέσης αρχικού πλανήτη
194     nbPlanets = len(planets)
195     start = random.randint(0, nbPlanets - 5)
196     # δημιουργία λίστας 4 διαδοχικών πλανητών
197     fourPlanets = planets[start : start + 4]
```

Θα δημιουργήσουμε ένα αντίγραφο της λίστας των τεσσάρων πλανητών, ώστε να τους ανακατέψουμε και να τους εμφανίσουμε στον παίκτη με τυχαία σειρά. Δεν ανακατεύουμε απευθείας την αρχική λίστα, τη χρειαζόμαστε για να υπολογίσουμε τη σωστή απάντηση.

```
198     # αντίγραφο της λίστας των 4 διαδοχικών πλανητών
199     shuffled = fourPlanets.copy()
200     # ανακάτεμα των 4 πλανητών σε τυχαίες θέσεις
201     random.shuffle(shuffled)
```

Η συνάρτηση `shuffle()` της βιβλιοθήκης `random` δέχεται σαν παράμετρο μια λίστα και ανακατατάσσει τις τιμές της σε τυχαίες θέσεις.

Τώρα μπορούμε να διατυπώσουμε την ερώτηση και να ζητήσουμε την απάντηση του παίκτη. Τα ονόματα των πλανητών θα εμφανίζονται αριθμημένα, ώστε ο παίκτης να γράφει τον αριθμό που αντιστοιχεί στον κάθε πλανήτη για να δώσει τη σωστή σειρά.

```
202     # ερώτηση στον παίκτη
203     print("Βάλε τους πλανήτες με τη σωστή σειρά:")
204     showMultiple(shuffled)
205     print("\nΔώσε 4 αριθμούς με κενά ανάμεσά τους.")
206     # η απάντηση "χωρίζεται" όπου υπάρχουν κενά
207     # και τα συστατικά της τοποθετούνται σε μια λίστα
208     answer = input().split()
```

Η μέθοδος `split()` εφαρμόζεται σε αλφαριθμητικές τιμές, τις οποίες χωρίζει σε επιμέρους τμήματα εκεί όπου υπάρχουν κενά και επιστρέφει μια λίστα με τα τμήματα αυτά.

Εδώ, η απάντηση του χρήστη θα έχει τη μορφή `"1 2 3 4"` και η `split()` θα παράγει μια λίστα της μορφής `["1", "2", "3", "4"]`.

Για να ελέγξουμε αν η απάντηση του παίκτη είναι σωστή, χρειάζεται να κατασκευάσουμε μια λίστα αποτελούμενη από τους αριθμούς των πλανητών *με τη σειρά που πρέπει να τους δώσει ο παίκτης*.

Για να βρούμε τους αριθμούς που πρέπει να πληκτρολογήσει ο παίκτης πρέπει να διατρέξουμε την αρχική λίστα των πλανητών και για κάθε πλανήτη να υπολογίσουμε τη θέση του στην ανακατεμένη λίστα. Για παράδειγμα: ποιός είναι ο πρώτος αριθμός που πρέπει να πληκτρολογήσει ο παίκτης; Είναι ο αριθμός που έχει ο πρώτος πλανήτης της αρχικής λίστας μέσα στην ανακατεμένη λίστα.

```
209     # λίστα που θα περιέχει την σωστή απάντηση
210     correct = []
211     # για κάθε έναν από τους 4 διαδοχικούς πλανήτες
212     for planet in fourPlanets:
213         # σειρά του πλανήτη στην ανακατεμένη λίστα
214         position = shuffled.index(planet) + 1
215         # προσθήκη σωστής σειράς στην απάντηση
216         correct.append(str(position))
```

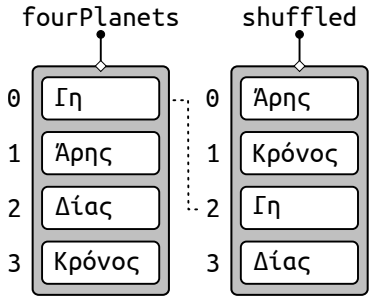
Ας ελέγξουμε πλέον αν η απάντηση του παίκτη είναι σωστή.

```
217     # έλεγχος απάντησης
218     if answer == correct:
219         print("Μπράβο, η σειρά είναι σωστή.")
220     else:
221         print("Η σωστή σειρά είναι:")
222         print(correct)
```

Ολοκληρώνουμε τα ερωτήματα που θα εμφανίσουμε στον παίκτη, καλώντας την συνάρτηση απ' το κύριο πρόγραμμα.

```
245 # εύρεση της σωστής σειράς των πλανητών
246 print("\nΕρώτηση 6:")
247 outOfOrder(planets)
```

src/planets.10.py



Σχήμα 5.9: Ένα παράδειγμα. Η Γη είναι πρώτη στην αρχική λίστα και τρίτη στην ανακατεμένη λίστα. Άρα ο πρώτος αριθμός που πρέπει να πληκτρολογήσει ο χρήστης είναι το 3.

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*.

Η συνάρτηση `str()` μετατρέπει μια τιμή σε αλφαριθμητική. Εδώ είναι απαραίτητο να μετατρέψουμε τις θέσεις των πλανητών σε αλφαριθμητικές για να συγκριθούν με τις αντίστοιχες αλφαριθμητικές τιμές που θα διαβαστούν από το πληκτρολόγιο.

Ο τελεστής == μπορεί να χρησιμοποιηθεί και για να συγκριθούν δύο λίστες, στοιχείο προς στοιχείο.

Τροποποιήσεις – Επεκτάσεις

Στο κεφάλαιο αυτό μάθαμε πολλά για τις λίστες και τους διάφορους τρόπους με τους οποίους μπορούμε να τις επεξεργαστούμε στην Python. Για να είναι ευκολότερο για εσάς να ανατρέχετε σε όλα αυτά καθώς λύνετε τις ασκήσεις, τα συγκεντρώσαμε όλα (και μερικά ακόμα) σ' ένα "σκονάκι".

pythonies.mysch.gr/cslists.pdf

5.1 *Υλοποιήστε τη συνάρτηση `addPluto()` έτσι ώστε να επιστρέφει την τιμή `True` όταν η απάντηση του χρήστη περιέχεται σε μια λίστα πιθανών καταφατικών απαντήσεων.*

Μπορείτε να εφαρμόσετε τη μέθοδο `lower()` στην απάντηση του χρήστη, οπότε η λίστα των πιθανών καταφατικών απαντήσεων θα είναι μικρότερη αφού θα πρέπει να περιέχει μόνο απαντήσεις με πεζά γράμματα.

5.2 *Υλοποιήστε τη συνάρτηση* `select()`*, έτσι ώστε η λίστα των πιθανών απαντήσεων να κατασκευάζεται επαναληπτικά ως εξής: σε κάθε βήμα θα επιλέγεται τυχαία μια απάντηση και θα προστίθεται στη λίστα με τις πιθανές απαντήσεις μέχρι να συμπληρωθεί ο απαραίτητος αριθμός. Θα πρέπει να εξασφαλίζεται ότι η σωστή απάντηση περιλαμβάνεται στην τελική λίστα και να γίνεται ο απαραίτητος έλεγχος, ώστε να μην προστίθεται πολλές φορές ο ίδιος πλανήτης στις πιθανές απαντήσεις.*

Διακρίνετε κάποιο μειονέκτημα σε μια τέτοια υλοποίηση;

5.3 *Υλοποιήστε τη συνάρτηση* `select()`*, έτσι ώστε να ανακατεύεται ένα αντίγραφο της αρχικής λίστας των απαντήσεων και μετά να επιλέγεται από αυτό ένα τμήμα με τη λειτουργία του τεμαχισμού, ως λίστα πιθανών απαντήσεων. Θα πρέπει να εξασφαλίζεται ότι η σωστή απάντηση περιλαμβάνεται στην τελική λίστα.*

Διακρίνετε κάποιο μειονέκτημα σε μια τέτοια υλοποίηση;

5.4 *Μπορείτε να φτιάξετε ένα παιχνίδι με ερωτήσεις πολλαπλής επιλογής που θα αφορούν τις ημέρες της εβδομάδας, αντί για τους πλανήτες, και θα απευθύνεται σε παιδιά πρώτης δημοτικού.*

Μερικές ενδεικτικές ερωτήσεις: Ποιά από τις παρακάτω μέρες *δεν* πηγαίνουμε σχολείο; Ποια μέρα είναι πριν ή μετά τη x; Αν σήμερα η μέρα είναι x, ποια ημέρα εννοούμε όταν λέμε "αύριο", "μεθαύριο", "χθες" ή "προχθές"; Αν σήμερα η μέρα είναι x, τι μέρα θα είναι σε n μέρες; Αν σήμερα η μέρα είναι y, πόσες ημέρες έχουν περάσει από τη x; Βάλτε στη σειρά τις παρακάτω ημέρες της εβδομάδας.

Εννοείται πως τα στοιχεία των ερωτήσεων που συμβολίζονται με x, y και n, όπως και οι πιθανές απαντήσεις, θα επιλέγονται κάθε φορά τυχαία.

5.5 *Μπορείτε να φτιάξετε ένα παιχνίδι με παρόμοιες ερωτήσεις πολλαπλής επιλογής που θα αφορούν τους μήνες και τις εποχές, αντί για τους πλανήτες, και θα απευθύνεται σε παιδιά πρώτης και δευτέρας δημοτικού.*

Μερικές ενδεικτικές ερωτήσεις: Ποιός είναι ο n-οστός μήνας του χρόνου; Ποιος αριθμός αντιστοιχεί στον μήνα x; Ποιος μήνας έρχεται πριν ή μετά τον μήνα x; Ποιος μήνας είναι πιο κοντά στην αρχή ή στο τέλος της χρονιάς; Σε ποια εποχή ανήκει ο μήνας x; Ποιος μήνας δεν ανήκει στην εποχή y; Ποια εποχή δεν "εκπροσωπείται" στους παρακάτω μήνες;

Εννοείται πως τα στοιχεία των ερωτήσεων που συμβολίζονται με x, y και n, όπως και οι πιθανές απαντήσεις, θα επιλέγονται κάθε φορά τυχαία.

5.6 *Να τροποποιήσετε το πρόγραμμα που παίζει Πέτρα – Ψαλίδι – Χαρτί έτσι ώστε να χρησιμοποιεί μια λίστα που περιέχει τις λέξεις* `"Πέτρα"`*,* `"Ψαλίδι"` *και* `"Χαρτί"`*. Έτσι, οι (ακέραιες) επιλογές του παίκτη και του προγράμματος θα αντιστοιχίζονται άμεσα σε μια από αυτές τις λέξεις, χωρίς την ανάγκη να χρησιμοποιηθεί η* `if`*.*

Ασκήσεις

5.7 *Τα μπισκότα τύχης (fortune cookies) είναι μπισκότα που στο εσωτερικό τους περιέχουν μηνύματα με προβλέψεις γι' αυτόν που τα ανοίγει. Υλοποιήστε ένα πρόγραμμα που θα καλεί τον χρήστη να ανοίξει ένα τυχερό μπισκότο και στη συνέχεια θα του εμφανίζει μια τυχαία πρόβλεψη.*

Μερικές ιδέες για τις προφητείες των μπισκότων: Η τύχη σου κρύβεται σε άλλο μπισκότο. Ο καλύτερος τρόπος να παραμείνεις υγιής είναι να συνεχίσεις να τρως τυχερά μπισκότα. Αποδέξου ότι κάποιες μέρες είσαι το περιστέρι και κάποιες άλλες το άγαλμα. Μαθαίνεις από τα λάθη σου... Σήμερα είναι μια μέρα που θα διδαχθείς πολλά. Η σκληρή δουλειά θα σου ανταποδώσει στο μέλλον. Η τεμπελιά θα σου ανταποδώσει άμεσα... Όλοι συμφωνούν: Είσαι ο καλύτερος!

5.8 *Στο παιχνίδι της αντιστροφής, εμφανίζεται στον παίκτη μια λίστα με αριθμούς κι εκείνος καλείται να τους αναδιατάξει σε αύξουσα σειρά. Το μόνο πράγμα που καθορίζει ο παίκτης σε κάθε κίνησή του είναι πόσοι αριθμοί θ' αντιστραφούν (ξεκινώντας από τ' αριστερά).*

Για παράδειγμα, αν οι αριθμοί είναι:

```
2 3 4 5 1 6 7 8 9
```

και ο παίκτης επιλέξει ν' αντιστρέψει τέσσερις αριθμούς, τότε οι αριθμοί θα γίνουν:

```
5 4 3 2 1 6 7 8 9
```

Αν στη συνέχεια ο παίκτης αντιστρέψει πέντε αριθμούς, κερδίζει!

```
1 2 3 4 5 6 7 8 9
```

Να κατασκευάσετε πρόγραμμα που εμφανίζει στον παίκτη μια λίστα με τους αριθμούς από το 1 μέχρι το 9 ανακατεμένους και στη συνέχεια ζητά από τον παίκτη επαναληπτικά πόσους αριθμούς να αντιστρέψει. Το παιχνίδι τελειώνει όταν ο παίκτης καταφέρει να τοποθετήσει τους αριθμούς στη σωστή σειρά ή όταν παραιτηθεί, απαντώντας ότι επιθυμεί να αντιστρέψει μηδέν αριθμούς. Αν ο παίκτης καταφέρει να ολοκληρώσει το παιχνίδι, το πρόγραμμα θα πρέπει να εμφανίζει πόσες κινήσεις χρειάστηκαν.

Όταν ολοκληρώσετε το πρόγραμμά σας, προσπαθήστε να το τροποποιήσετε έτσι ώστε οι κινήσεις να μην επιλέγονται από το χρήστη, αλλά από το πρόγραμμα, το οποίο πια θα παίζει μόνο του.

Η μέθοδος `reverse()` εφαρμόζεται σε μια λίστα κι αντιστρέφει τα στοιχεία της. Για παράδειγμα, η εντολή `L.reverse()` αντιστρέφει τη λίστα `L`.

Ο *τεμαχισμός* μπορεί επίσης να χρησιμοποιηθεί για την αντιστροφή μιας λίστας. Για παράδειγμα, η έκφραση `L[::-1]` δημιουργεί μια νέα λίστα, που περιέχει όλα τα στοιχεία της `L` με την αντίστροφη σειρά.

Εδώ χρειάζεται ν' αντιστρέψετε μόνο ένα τμήμα της λίστας, οπότε θα χρειαστεί πρώτα να την *τεμαχίσετε* και, μετά την αντιστροφή, να συνενώσετε τα τμήματά της.

5.9 *Ο Ηλεκτρικός της Αθήνας εγκαινιάστηκε το 1869 και συνέδεε τον Πειραιά με το Θησείο. Μετά από 88 χρόνια η διαδρομή επεκτάθηκε μέχρι την Κηφισιά. Σήμερα αριθμεί 24 σταθμούς.*

Υλοποιήστε ένα πρόγραμμα που θα βοηθά το χρήστη να μετακινηθεί με τον Ηλεκτρικό. Ο χρήστης θα δίνει το όνομα του σταθμού από τον οποίο θ' αναχωρήσει και το όνομα του σταθμού στον οποίο θέλει να φτάσει και το πρόγραμμα θα του εμφανίζει την κατεύθυνση που θα ακολουθήσει (προς Πειραιά ή προς Κηφισιά) και τα ονόματα όλων των ενδιάμεσων σταθμών της διαδρομής.

Βρείτε τη λίστα με τους σταθμούς στο αρχείο `pythonies.mysch.gr/src/metro.py`. Αν το πρόγραμμά σας βρίσκεται στον ίδιο κατάλογο με αυτό το αρχείο μπορείτε να γράψετε `from metro import stations` και από εκεί και πέρα θα μπορείτε να χρησιμοποιήσετε τη λίστα `stations` με τους 24 σταθμούς.

Ένα παράδειγμα αλληλεπίδρασης με το πρόγραμμα:

```
Δώστε σταθμό αφετηρίας
ΠΕΤΡΑΛΩΝΑ
Δώστε σταθμό προορισμού
ΦΑΛΗΡΟ
Σταθμοί από ΠΕΤΡΑΛΩΝΑ προς ΦΑΛΗΡΟ με κατεύθυνση ΠΕΙΡΑΙΑΣ
ΤΑΥΡΟΣ
ΚΑΛΛΙΘΕΑ
ΜΟΣΧΑΤΟ
```

Μετά την εμφάνιση των πληροφοριών, το πρόγραμμα θα ρωτάει το χρήστη αν επιθυμεί πληροφορίες για κάποια άλλη διαδρομή και θα επαναλαμβάνει τη διαδικασία σε περίπτωση καταφατικής απάντησης.

5.10 *Το τρίγωνο του Pascal είναι ένας τριγωνικός μαθηματικός πίνακας με πολύ ενδιαφέρουσες ιδιότητες. Για να το κατασκευάσουμε, τοποθετούμε αρχικά στην κορυφή του τριγώνου, δηλαδή στη γραμμή 0, τον αριθμό 1. Κάθε αριθμός σε κάθε επόμενη γραμμή προκύπτει ως άθροισμα των δύο αριθμών που βρίσκονται* από πάνω του*, εκτός από τον πρώτο και τον τελευταίο αριθμό κάθε γραμμής που είναι πάντα το 1.*

$n = 0$	1						
$n = 1$	1	1					
$n = 2$	1	2	1				
$n = 3$	1	3	3	1			
$n = 4$	1	4	6	4	1		
$n = 5$	1	5	10	10	5	1	
$n = 6$	1	6	15	20	15	6	1

Σχήμα 5.10: Οι 7 πρώτες γραμμές του τριγώνου του Pascal.

Από το τρίγωνο του Pascal μπορούμε να εξάγουμε τους τριγωνικούς αριθμούς και τους αριθμούς Fibonacci, να υπολογίσουμε το ανάπτυγμα πολυωνύμων ακόμα και να κατασκευάσουμε ένα φράκταλ που ονομάζεται τρίγωνο Sierpinski.

Μια ιδιότητα του τριγώνου που θα χρησιμοποιήσουμε εμείς είναι η εξής: ο αριθμός που βρίσκεται στη γραμμή n και στη στήλη y αντιστοιχεί στο πλήθος των διαφορετικών τρόπων με τους οποίους μπορούμε να φέρουμε y φορές κορώνα αν στρίψουμε ένα νόμισμα n φορές (θυμηθείτε ότι οι γραμμές και οι στήλες αριθμούνται από το 0). Διαιρώντας αυτόν τον αριθμό με το άθροισμα των αριθμών της γραμμής n, το οποίο είναι 2^n, υπολογίζουμε την πιθανότητα να φέρουμε y φορές κορώνα αν στρίψουμε ένα νόμισμα n φορές.

Να υλοποιήσετε ένα πρόγραμμα το οποίο διαβάζει το n από το χρήστη, κατασκευάζει το τρίγωνο του Pascal μέχρι και τη γραμμή n και εμφανίζει, για κάθε y, την πιθανότητα να φέρουμε y φορές κορώνα αν στρίψουμε ένα νόμισμα n φορές.

5.11 *Στο τυχερό παιχνίδι* Λοττο*, ο παίκτης επιλέγει έξι αριθμούς από το 1 μέχρι και το 49. Στη συνέχεια κληρώνονται έξι αριθμοί με τυχαίο τρόπο.*

Ανάλογα με το πλήθος των αριθμών που κατάφερε να μαντέψει ο παίκτης, έχει και τα αντίστοιχα κέρδη.

Να υλοποιήσετε πρόγραμμα που θα ζητάει από τον χρήστη τους έξι αριθμούς της επιλογής του. Στη συνέχεια θα παράγει τους έξι τυχερούς αριθμούς που κληρώνονται. Θα πρέπει να εξασφαλίζεται ότι δεν εμφανίζεται πολλές φορές ο ίδιος αριθμός είτε στην εξάδα των τυχερών αριθμών είτε στην εξάδα των αριθμών που επιλέγει ο παίκτης. Το πρόγραμμα θα εμφανίζει στον παίκτη τους αριθμούς που επέλεξε, τους τυχερούς αριθμούς και πόσους από τους τυχερούς αριθμούς πέτυχε σωστά.

Η συνάρτηση `sorted()` δέχεται σαν παράμετρο μια λίστα και επιστρέφει μια νέα *ταξινομημένη* λίστα, χωρίς να πειράζει την αρχική. Η μέθοδος `sort()` εφαρμόζεται σε μια λίστα και ταξινομεί τα στοιχεία της. Με τα παραπάνω θα μπορέσετε να εμφανίζετε στο χρήστη τις εξάδες των αριθμών ταξινομημένες, όπως συνηθίζεται.

Καλό είναι να υλοποιήσετε τις επιμέρους λειτουργίες του προγράμματος με συναρτήσεις. Θα μπορούσατε να υλοποιήσετε τις εξής:

- `inputNumbers()` που διαβάζει από το χρήστη 6 αριθμούς, εξασφαλίζοντας ότι βρίσκονται μεταξύ `1` και `49` και δεν επαναλαμβάνονται, και τους επιστρέφει σε μια λίστα.
- `generateNumbers()` που δημιουργεί 6 τυχαίους αριθμούς μεταξύ `1` και `49`, εξασφαλίζοντας ότι δεν επαναλαμβάνονται, και τους επιστρέφει σε μια λίστα.
- `compare()` που δέχεται σαν παραμέτρους δύο λίστες και επιστρέφει μια λίστα με τα στοιχεία που βρίσκονται και στις δύο. Εναλλακτικά, θα μπορούσε να επιστρέφει απλά το πλήθος αυτών των στοιχείων.

5.12 *Το παιχνίδι Blackjack είναι ένα παιχνίδι με χαρτιά που παίζεται από δύο παίκτες. Ένας από τους δύο κάνει τη "μάνα" και είναι εκείνος που διαχειρίζεται την τράπουλα. Η τράπουλα περιέχει δεκατρία χαρτιά που επαναλαμβάνονται τέσσερις φορές (σύνολο 52): τα χαρτιά από 1 έως 10 και τις φιγούρες του Βαλέ, της Ντάμας και του Ρήγα. Η αξία του Βαλέ, της Ντάμας και του Ρήγα είναι 10 πόντοι, ενώ η αξία του Άσσου μπορεί να υπολογιστεί είτε ως 1 είτε ως 11, ανάλογα με το συμφέρον του παίκτη.*

Μπορείτε να αναπαραστήσετε την τράπουλα σαν λίστα με 52 στοιχεία. Για να κατασκευάσετε αυτή τη λίστα, μπορείτε να ξεκινήσετε με μια κενή λίστα, στην οποία θα προσθέτετε επαναληπτικά τα στοιχεία με την μέθοδο `append()`. Εναλλακτικά, με την έκφραση `52 * [None]`, μπορείτε να ξεκινήσετε με μια λίστα 52 στοιχείων χωρίς τιμή (αυτό είναι το `None`) και να προσδιορίσετε επαναληπτικά την τιμή κάθε στοιχείου. Μπορείτε ακόμα και να κατασκευάσετε μια λίστα `L`, που θα περιέχει μόνο τα 13 μοναδικά χαρτιά της τράπουλας, και μετά να κατασκευάσετε ολόκληρη την τράπουλα με την έκφραση `L * 4`.

Αρχικά, η τράπουλα ανακατεύεται και η "μάνα" με τον παίκτη παίρνουν από δύο χαρτιά ο καθένας: τα χαρτιά του παίκτη είναι "ανοικτά", δηλαδή φαίνεται η ένδειξή τους, ενώ η "μάνα" έχει ένα χαρτί "ανοικτό" και κρατά το άλλο "κλειστό". Η "μάνα" ρωτάει τον παίκτη αν επιθυμεί κι άλλο χαρτί, διαδικασία που επαναλαμβάνεται όσο ο παίκτης απαντά καταφατικά και η συνολική αξία των χαρτιών του είναι μικρότερη από 21. Στη συνέχεια, εφόσον το άθροισμα των χαρτιών του παίκτη δεν ξεπερνά το 21, η μάνα αποκαλύπτει το "κλειστό" της χαρτί και επαναλαμβάνει τη διαδικασία για τον εαυτό της. Αν κάποιος από τους δύο παίκτες ξεπεράσει το 21 τότε "καίγεται" και χάνει αυτομάτως. Σε διαφορετική περίπτωση, νικητής είναι εκείνος του οποίου τα χαρτιά έχουν το μεγαλύτερο άθροισμα, με τη "μάνα" να θεωρείται νικήτρια σε περίπτωση ισοβαθμίας.

Υλοποιήστε ένα πρόγραμμα το οποίο θα έχει το ρόλο της "μάνας" και θα παίζει Blackjack με αντίπαλο το χρήστη.

Υπόδειξη: Όσους Άσσους κι αν έχει ένας παίκτης, μόνο ένας από αυτούς μπορεί να μετρήσει ως 11. Επομένως, το πρόγραμμά σας θα πρέπει να ελέγχει το πολύ δύο αθροίσματα, το κανονικό άθροισμα των χαρτιών και, στην περίπτωση που υπάρχει τουλάχιστον ένας Άσσος, το λεγόμενο "μαλακό" άθροισμα, στο οποίο ο Άσσος μετράει ως 11.

Δομες Δεδομενων και Λιστες *Οι λίστες, δομές απλές κι ευέλικτες, έχουν βαρύνουσα σημασία στην ιστορία της Πληροφορικής. Η δεύτερη γλώσσα προγραμματισμού που αναπτύχθηκε, η LISP του John McCarthy, βασίζεται στις λίστες. Η Logo, με την σειρά της, μια ιστορική εκπαιδευτική γλώσσα προγραμματισμού, βασίζεται κυρίως στη LISP.*

Οι λίστες είναι ένα παράδειγμα οργάνωσης *των δεδομένων μας, έτσι ώστε να αποτελούν μια ενιαία συλλογή με συγκεκριμένη* δομή. *Υπάρχουν αναρίθμητοι τρόποι να οργανώσουμε τα δεδομένα μας, γι' αυτό και υπάρχουν και πάρα πολλές δομές δεδομένων. Ακόμα και οι πιο θεμελιώδεις έχουν εξωτικά ονόματα όπως* στοίβες, ουρές, δέντρα, σωροί *ή* γράφοι. *Ο τρόπος οργάνωσης καθορίζει πόσο εύκολο είναι να επεξεργαστούμε τα δεδομένα μας με συγκεκριμένο τρόπο. Γι' αυτό και επιλέγουμε τη δομή που θα χρησιμοποιήσουμε κάθε φορά ανάλογα με τον συγκεκριμένο τρόπο που θέλουμε να επεξεργαστούμε τα δεδομένα μας, στα πλαίσια ενός συγκεκριμένου προβλήματος.*

Μέχρι να αντιμετωπίσει κανείς στοιχειωδώς περίπλοκα προβλήματα είναι δύσκολο να διακρίνει γιατί *χρειάζεται να οργανωθούν τα δεδομένα – οι μεμονωμένες μεταβλητές φαίνονται υπεραρκετές, αλλά δεν είναι. Αν είστε άνθρωποι με περιπετειώδες πνεύμα, μπορείτε να επιχειρήσετε να υλοποιήσετε το παιχνίδι αυτού του κεφαλαίου χωρίς να χρησιμοποιήσετε λίστες.*

Αριθμηση απο το Μηδεν *Υπάρχουν δύο ειδών άνθρωποι στον κόσμο: (1) αυτοί που ξεκινούν την αρίθμηση από το ένα και (1) αυτοί που ξεκινούν την αρίθμηση από το μηδέν. Αν πιάσατε το αστειάκι, έχετε καταλάβει αυτό το λεπτό σημείο του κεφαλαίου. Η χρήση της αρίθμησης από το μηδέν ξεκίνησε τη δεκαετία του '60 από ανάγκη, για καθαρά τεχνικούς λόγους και είναι αλήθεια ότι σε εκείνους που την συναντούν για πρώτη φορά φαίνεται από άβολη έως παρανοϊκή. Ωστόσο, έχει και σημαντικά πλεονεκτήματα. Ο ιδιόρυθμος Edsger Dijkstra, ένας από τους ανθρώπους που συνέβαλαν καθοριστικά στην καθιέρωση της Πληροφορικής ως επιστήμη, έγραψε το 1982 τρεις ολόκληρες (χειρόγραφες) σελίδες με τίτλο "Γιατί η αρίθμηση πρέπει να ξεκινά από το μηδέν". Τώρα, η αρίθμηση από το μηδέν είναι πια από τα πολύ χαρακτηριστικά ιδιώματα της Πληροφορικής. Στο παιδικό βιβλίο "Lauren Ipsum" του Carlos Bueno, η αρίθμηση των κεφαλαίων ξεκινά από το μηδέν και η ηρωίδα, καθώς αρχίζει την περιπέτειά της, συναντά μια πινακίδα όπου το πρώτο μίλι της διαδρομής είναι φυσικά... το μηδενικό.*

Σχήμα 5.11: Από το Κεφάλαιο 0 του Lauren Ipsum.