Μπαρμπούτι

2

29 Αυγούστου 2016
11:35

Σε αυτό το κεφάλαιο θα φτιάξουμε ένα παιχνίδι με ζάρια. Στην πορεία θα εξασκηθούμε στη *δομή επιλογής* και θα γνωρίσουμε τη *δομή επανάληψης*, που μας επιτρέπει να εκτελούμε τις ίδιες εντολές πολλές φορές. Το τελικό πρόγραμμα θα είναι μικρό, όμως σύντομα θα χρειαστεί να γράψουμε προγράμματα μεγαλύτερα και πιο περίπλοκα. Έτσι, το ουσιαστικό αντικείμενο του κεφαλαίου είναι να έρθουμε σε επαφή με τον τρόπο που οι προγραμματιστές συνθέτουν τα προγράμματά τους από *υποπρογράμματα*, σα να συναρμολογούν ένα περίπλοκο μηχάνημα από απλούστερα εξαρτήματα.

Έννοιες: δομή επιλογής, δομή επανάληψης, υποπρογράμματα

Σε πολλές αμερικάνικες ταινίες οι πρωταγωνιστές κάνουν μια βόλτα από το Λας Βέγκας και γίνονται εκατομμυριούχοι ή χάνουν τα πάντα παίζοντας ένα παιχνίδι με ζάρια, άγνωστο στους περισσότερους. Η κοντινότερη ελληνική εκδοχή αυτού του παιχνιδιού είναι το μπαρμπούτι, πηγή έμπνευσης για πολλούς άτυχους ρεμπέτες.

Οι κανόνες είναι οι εξής: Αρχικά, ο παίκτης ρίχνει δύο ζάρια. Αν το άθροισμα των ενδείξεών τους είναι 7 ή 11, τότε ο παίκτης κερδίζει, ενώ αν είναι 2, 3 ή 12 χάνει. Σε οποιαδήποτε άλλη περίπτωση, το άθροισμα των ενδείξεων γίνεται ο "στόχος" και ο παίκτης ρίχνει *επαναλαμβανόμενα* τα ζάρια μέχρι να ρίξει την ίδια ζαριά (να πετύχει τον στόχο), οπότε και κερδίζει, ή μέχρι να φέρει 7, οπότε και χάνει.

Τα Κόκκαλα Στον Μάστορα

Πώς θα προσομοιώσουμε το ρίξιμο των ζαριών;

Θα ασχοληθούμε αρχικά με την πρώτη ζαριά του παίκτη, με την οποία ξεκινά το παιχνίδι. Από την κατάλληλη *βιβλιοθήκη* θα χρησιμοποιήσουμε έναν μηχανισμό παραγωγής τυχαίων αριθμών.

Για να χρησιμοποιήσουμε μια βιβλιοθήκη θα πρέπει πρώτα να την εισάγουμε (`import`). Εδώ θα εισάγουμε τη βιβλιοθήκη `random` και θα χρησιμοποιήσουμε την `randint()`, που παράγει τυχαίους ακέραιους εντός συγκεκριμένων ορίων που καθορίζει ο προγραμματιστής.

```
1 import random
2 # τυχαίες τιμές για τα δύο ζάρια
3 dice1 = random.randint(1,6)
4 dice2 = random.randint(1,6)
```

Οι μεταβλητές `dice1` και `dice2` αντιστοιχούν στις ενδείξεις των δύο ζαριών και παίρνουν μια τυχαία τιμή από το 1 μέχρι και το 6. Κάθε φορά που θα εκτελείται το πρόγραμμα η τιμή αυτή θα είναι (πιθανώς) διαφορετική.

Μένει τώρα να εμφανίσουμε τη ζαριά στο χρήστη, αφού πρώτα υπολογίσουμε το άθροισμα των δύο ενδείξεων `dice1` και `dice2`.

```
5  # υπολογισμός και εμφάνιση ζαριάς
6  roll = dice1 + dice2
7  print("Έριξες", dice1, dice2, "=", roll)
```

craps/src/craps.1.py

Η εντολή `roll = dice1 + dice2` σημαίνει: υπολόγισε το άθροισμα των τιμών `dice1` και `dice2` κι ονόμασε το αποτέλεσμα `roll`. Αυτό θα συμβεί *μια φορά*, όταν εκτελεστεί η εντολή, και θα χρησιμοποιηθούν οι τιμές που έχουν οι μεταβλητές `dice1` και `dice2` *εκείνη την στιγμή*. Αν οι `dice1` και `dice2` αργότερα αλλάξουν τιμή, *δεν* θα κάνει το ίδιο αυτόματα και η `roll`. Η `roll` θα αλλάξει τιμή μόνο όταν της αποδοθεί ρητά, με ένα `=`, μια (οποιαδήποτε) άλλη τιμή.

Όπως έχει τώρα το πρόγραμμα, τα ζάρια ρίχνονται αμέσως μόλις ξεκινήσει το παιχνίδι. Είναι όμως καλύτερο ν' αποφασίζει ο παίκτης πότε θα ριχτεί η ζαριά, για να του δώσουμε την αίσθηση ότι παίζει.

```
2  # προτροπή για ρίψη ζαριών
3  print("Ρίξε τα ζάρια με ENTER...", end="")
4  input()
5  # τυχαίες τιμές για τα δύο ζάρια
6  dice1 = random.randint(1,6)
7  dice2 = random.randint(1,6)
```

Εδώ η τιμή που επιστρέφει η `input()`, δηλαδή το κείμενο που πληκτρολογεί ο χρήστης, δεν αποδίδεται σε κάποια μεταβλητή και χάνεται, αφού δεν μας ενδιαφέρει να τη διατηρήσουμε.

Για να μην αλλάξει η γραμμή μετά την εμφάνιση της προτροπής, ορίζουμε την παράμετρο `end` της `print()` να είναι ίση με το κενό κείμενο.

Τώρα τα ζάρια ρίχνονται αφού πρώτα ο παίκτης προτρέπεται να πατήσει το πλήκτρο ENTER.

Πάλι Ντόρτια Ήφερα

Και πώς θα ξέρουμε αν ο παίκτης κέρδισε, έχασε ή πρέπει να ξαναρίξει;

Στο σημείο αυτό, υπάρχουν τρεις διαφορετικές περιπτώσεις, τρεις πιθανές εκδοχές για την συνέχεια του παιχνιδιού: νίκη (με 7 ή 11), ήττα (με 2, 3 ή 12) ή επανάληψη (σε οποιαδήποτε άλλη περίπτωση). Σε κάθε περίπτωση είναι διαφορετικές οι εντολές που θα πρέπει να εκτελεστούν.

Κατά τη διάρκεια συγγραφής του προγράμματος, δεν είναι δυνατό να γνωρίζουμε εκ των προτέρων ποια θα είναι κάθε φορά η ζαριά, οπότε θα χρησιμοποιήσουμε μια *δομή επιλογής*, με την οποία θα καθορίσουμε ποιες εντολές θα πρέπει να εκτελεστούν σε κάθε μία από τις τρεις πιθανές περιπτώσεις.

```
# έλεγχος αποτελέσματος
if roll == 7 or roll == 11:
    # νίκη με την πρώτη
    print("Κέρδισες με την πρώτη!")
elif roll <= 3 or roll == 12:
    # ήττα με την πρώτη
    print("Έχασες με την πρώτη...")
else:
    # τίθεται ο "στόχος"
    print("Ξαναρίξε. Πρέπει να φέρεις", roll)
```

Κατά την εκτέλεση του προγράμματος ελέγχονται διαδοχικά οι συνθήκες, η μία μετά την άλλη, μέχρι να βρεθεί μία που να είναι αληθής. Όταν συμβεί αυτό, εκτελούνται οι αντίστοιχες εντολές και δεν ελέγχεται άλλη συνθήκη. Αν διαπιστωθεί ότι όλες οι συνθήκες είναι ψευδείς, εκτελούνται οι εντολές της **else**, αν υπάρχουν. Επομένως, τελικά θα εκτελεστούν *μόνο* είτε οι εντολές που αντιστοιχούν στην πρώτη αληθή συνθήκη, είτε οι εντολές της **else**, είτε καμία εντολή.

Ξαναρίχνοντας τα Ζάρια

Η τρίτη περίπτωση δεν είναι ολοκληρωμένη. Αν ο παίκτης δεν κερδίσει, ούτε χάσει με την πρώτη, πρέπει να ξαναρίξει τα ζάρια.

Το πρόγραμμα περιέχει *ήδη* ένα σύνολο εντολών που "ρίχνουν" τα δύο ζάρια. Τις έχουμε χρησιμοποιήσει για την πρώτη ζαριά και μπορούμε να τις αντιγράψουμε και στην περίπτωση όπου ο παίκτης πρέπει να ξαναρίξει τα ζάρια.

```
else:
    # τίθεται ο "στόχος"
    print("Ξαναρίξε. Πρέπει να φέρεις", roll)
    # προτροπή για ρίψη ζαριών
    print("Ρίξε τα ζάρια με ENTER...", end="")
    input()
    # τυχαίες τιμές για τα δύο ζάρια
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    # υπολογισμός και εμφάνιση ζαριάς
    roll = dice1 + dice2
    print("Έριξες", dice1, dice2, "=", roll)
```

Κάποια στιγμή θα πρέπει να ελεγχθεί αν η δεύτερη ζαριά του παίκτη είναι ίση με την πρώτη, για να διαπιστωθεί αν κέρδισε ή όχι. Το πρόβλημα είναι ότι, με το πρόγραμμα ως έχει, αυτό είναι αδύνατο: Όταν ρίχνεται η δεύτερη ζαριά, το αποτέλεσμά της ονομάζεται και πάλι `roll`. Η προηγούμενη τιμή της `roll`, η τιμή της πρώτης ζαριάς, δεν είναι πια διαθέσιμη και δεν μπορεί να γίνει σύγκριση μεταξύ των δύο τιμών (δείτε σχετικά και τα σχήματα 2.1 και 2.2).

Κάθε **if** συνοδεύεται από μια *συνθήκη*, η οποία ελέγχεται κατά την εκτέλεση του προγράμματος και μπορεί να είναι αληθής (**True**) ή ψευδής (**False**).

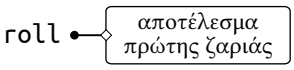
Το **elif** σημαίνει **else if**. Μετά από μια **if** μπορούμε να χρησιμοποιήσουμε όσες **elif** είναι απαραίτητες.

Οι εντολές που ακολουθούν τις **if** και **elif** είναι στοιχισμένες δεξιότερα. Η στοίχιση υποδηλώνει ότι αυτές οι εντολές θα εκτελεστούν *μόνο* αν η αντίστοιχη συνθήκη είναι αληθής και καμία από τις προηγούμενες συνθήκες δεν είναι αληθείς. Οι εντολές που ακολουθούν την **else** είναι επίσης στοιχισμένες δεξιότερα και θα εκτελεστούν μόνο εφόσον οι προηγούμενες συνθήκες είναι ψευδείς.

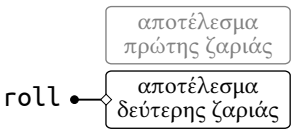
Μην παραλείπετε το σύμβολο : μετά τις συνθήκες και την **else**.

Με το == ελέγχεται αν δύο τιμές είναι ίσες. Διαφέρει από το = που χρησιμοποιείται για να δώσουμε τιμή σε μια μεταβλητή.

Το **or** χρησιμοποιείται για τη διάζευξη δύο συνθηκών. Η διάζευξη είναι αληθής όταν *τουλάχιστον μια* από τις διαζευγμένες συνθήκες είναι αληθής. Υπάρχει και ο τελεστής **and**. Χρησιμοποιείται για τη *σύζευξη* δύο συνθηκών, η οποία είναι αληθής όταν *και οι δύο* συζευγμένες συνθήκες είναι αληθείς.

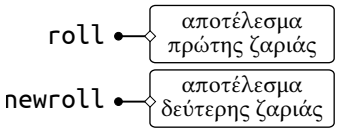


Σχήμα 2.1: Αρχικά, η τιμή της μεταβλητής `roll` αντιστοιχεί στο αποτέλεσμα της πρώτης ζαριάς.



Σχήμα 2.2: Στη συνέχεια, η `roll` χρησιμοποιείται για το αποτέλεσμα της δεύτερης ζαριάς και η τιμή της τροποποιείται. Το αποτέλεσμα της πρώτης ζαριάς δεν είναι πια διαθέσιμο.

Για να λυθεί το πρόβλημα θα χρησιμοποιήσουμε μια *διαφορετική μεταβλητή*, ένα διαφορετικό όνομα για τη δεύτερη ζαριά.



Σχήμα 2.3: Το αποτέλεσμα της δεύτερης ζαριάς ονομάζεται `newroll`. Οι δυο ζαριές, οι τιμές `roll` και `newroll` μπορούν να συγκριθούν μεταξύ τους.

```
18 else:
19     # τίθεται ο "στόχος"
20     print("Ξαναρίξε. Πρέπει να φέρεις", roll)
21     # προτροπή για ρίψη ζαριών
22     print("Ρίξε τα ζάρια με ENTER...", end="")
23     input()
24     # τυχαίες τιμές για τα δύο ζάρια
25     dice1 = random.randint(1,6)
26     dice2 = random.randint(1,6)
27     # υπολογισμός και εμφάνιση ζαριάς
28     newroll = dice1 + dice2
29     print("Έριξες", dice1, dice2, "=", newroll)
```

Τώρα μπορούμε να συγκρίνουμε τη δεύτερη ζαριά με την πρώτη, δηλαδή την τιμή της `newroll` με εκείνη της `roll` για να εξετάσουμε αν ο παίκτης κέρδισε ή όχι.

```
28     newroll = dice1 + dice2
29     print("Έριξες", dice1, dice2, "=", newroll)
30     # έλεγχος αποτελέσματος
31     if newroll == roll:
32         print("Κέρδισες!")
33     elif newroll == 7:
34         print("Έχασες...")
```

craps/src/craps.2.py

Ξανά Και Ξανά

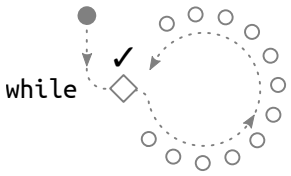
Η τρίτη περίπτωση ακόμα δεν είναι ολοκληρωμένη. Αν ο παίκτης δεν κερδίσει, ούτε χάσει με την πρώτη, ίσως χρειαστεί να ρίξει περισσότερες από μία επιπλέον ζαριές. Πώς θα κάνουμε τις υπάρχουσες εντολές για τη δεύτερη ζαριά να επαναλαμβάνονται;

Κάθε γλώσσα προγραμματισμού προσφέρει *επαναληπτικές δομές*, δηλαδή τρόπους να εκφράσει κανείς ότι ένα σύνολο εντολών θα πρέπει να επαναλαμβάνεται.

Μια τέτοια επαναληπτική δομή θα χρειαστεί να προσθέσουμε στην τρίτη περίπτωση, όπου ο παίκτης ούτε κερδίζει, ούτε χάνει με την πρώτη. *Μέσα* στην επανάληψη θα τοποθετήσουμε τις εντολές που υλοποιούν την ρίψη των ζαριών και ελέγχουν το αποτέλεσμα. Οι εντολές γράφονται μόνο μια φορά αλλά εκτελούνται ξανά και ξανά.

Στην Python, η βασική επαναληπτική δομή είναι η **`while`**, η οποία μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση, ενώ για συγκεκριμένου είδους επαναλήψεις υπάρχει και η δομή **`for`**.

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



Σχήμα 2.4: Η τετριμμένη συνθήκη **`True`** που ελέγχεται εδώ από τη **`while`** είναι πάντα αληθής, κι έτσι η συγκεκριμένη επανάληψη θα ξεκινήσει σίγουρα και δεν πρόκειται να διακοπεί λόγω της συνθήκης.

```
18 else:
19     # τίθεται ο "στόχος"
20     print("Ξαναρίξε. Πρέπει να φέρεις", roll)
21     # επανάληψη ρίψεων
22     while True:
```

```
23          # προτροπή για ρίψη ζαριών
24          print("Ρίξε τα ζάρια με ENTER...", end="")
25          input()
26          # τυχαίες τιμές για τα δύο ζάρια
27          dice1 = random.randint(1,6)
28          dice2 = random.randint(1,6)
29          # υπολογισμός και εμφάνιση ζαριάς
30          newroll = dice1 + dice2
31          print("Έριξες", dice1, dice2, "=", newroll)
```

Οι εντολές που ακολουθούν τη **while** είναι στοιχισμένες δεξιότερα. Η στοίχιση αυτή επιτυγχάνεται εισάγοντας κενά πριν από τις εντολές, τα οποία υποδηλώνουν ότι αυτές οι εντολές "ανήκουν" στη **while** και θα επαναλαμβάνονται. Η πρώτη εντολή μετά τη **while** που δεν θα είναι στοιχισμένη δεξιότερα δεν θα επαναλαμβάνεται, αλλά θα εκτελεστεί μόνο μια φορά, όταν η επανάληψη τερματιστεί.

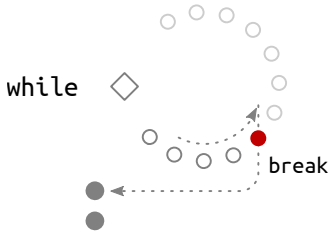
Είναι Σημαντικό Να Έχεις Στόχους

Οι επαναλαμβανόμενες ρίψεις των ζαριών πρέπει κάποτε να σταματούν. Πώς τερματίζω την επαναληπτική διαδικασία;

Ήδη το πρόγραμμά μας ελέγχει μετά από κάθε ζαριά την έκβασή της. Υπάρχουν κι εδώ τρεις πιθανές περιπτώσεις: νίκη (ο παίκτης φέρνει ζαριά ίση με τον στόχο), ήττα (ο παίκτης φέρνει 7) και συνέχεια της επανάληψης (σε οποιαδήποτε άλλη περίπτωση). Στις δύο πρώτες περιπτώσεις η επανάληψη θα πρέπει να τερματιστεί. Ένας απλός τρόπος για να επιτευχθεί αυτό είναι με την προσθήκη της εντολής **break**, η οποία διακόπτει άμεσα τον κύκλο της επανάληψης.

```
32          # έλεγχος αποτελέσματος
33          if newroll == roll:
34              print("Κέρδισες!")
35              # το παιχνίδι τελείωσε
36              break
37          elif newroll == 7:
38              print("Έχασες...")
39              # το παιχνίδι τελείωσε
40              break
```

craps/src/craps.3.py



Σχήμα 2.5: Η **break** διακόπτει και τερματίζει *άμεσα* τον κύκλο της επανάληψης, χωρίς να ελεγχθεί η συνθήκη συνέχειας. Οι εντολές μετά την **break** αγνοούνται και η εκτέλεση συνεχίζεται από την πρώτη εντολή που ακολουθεί την επανάληψη.

Για να κατανοήσετε καλύτερα τι σημαίνει η άμεση διακοπή της επανάληψης όταν εκτελεστεί η **break**, δοκιμάστε να τοποθετήσετε τις `print` *μετά* τις **break**. Θα διαπιστώσετε ότι τα μηνύματα δεν θα εμφανιστούν ποτέ.

Η **else** δεν είναι υποχρεωτική σε μια δομή επιλογής και παραλείπεται όταν δεν υπάρχουν εντολές να εκτελεστούν στην τελευταία περίπτωση.

Η χρήση της **break** είναι μια πρακτική που δεν ακολουθείται από όλους. Ορισμένοι θεωρούν ότι ο κώδικας είναι πιο κατανοητός όταν υπάρχει ένα μοναδικό σημείο εξόδου από την επανάληψη: η συνθήκη συνέχειας. Θα εξετάσουμε μια εκδοχή στην οποία ο κύκλος της επανάληψης δεν διακόπτεται άμεσα με την **break** αλλά μόνο όταν η συνθήκη συνέχειας ελεγχθεί και διαπιστωθεί ότι είναι ψευδής.

Θα χρησιμοποιήσουμε μια *λογική* μεταβλητή `over` για να "θυμάται" το πρόγραμμά μας αν το παιχνίδι έχει τελειώσει. Αρχικά, πριν την επανάληψη, η `over` ορίζεται ως ψευδής (**False**).

Υπάρχουν μόνο δύο *λογικές* τιμές: **True** ή **False**.

```
18  else:
19      # τίθεται ο "στόχος"
20      print("Ξαναρίξε. Πρέπει να φέρεις", roll)
21      # το παιχνίδι δεν έχει τελειώσει
22      over = False
```

Στην αρχή κάθε κύκλου της επανάληψης, ελέγχεται η συνθήκη συνέχειας `not over` κι ένας νέος κύκλος ξεκινά μόνο εφόσον η `over` εξακολουθεί να είναι ψευδής.

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

```
21      # το παιχνίδι δεν έχει τελειώσει
22      over = False
23      # επανάληψη ρίψεων
24      while not over:
```

Η συνθήκη συνέχειας της επανάληψης θα μπορούσε εναλλακτικά να γραφτεί και με τους δύο ακόλουθους τρόπους:

```
    # επανάληψη ρίψεων
    while over == False
```

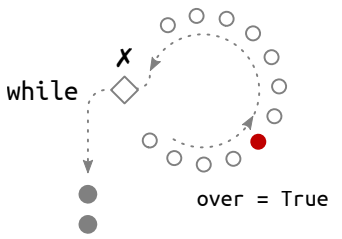
Με το `!=` ελέγχεται αν δύο τιμές είναι διαφορετικές.

```
    # επανάληψη ρίψεων
    while over != True
```

Η τιμή της `over` θα πρέπει να αλλάζει σε αληθής (`True`) όταν ο παίκτης κερδίσει ή χάσει – εκεί δηλαδή όπου στην προηγούμενη εκδοχή συναντούσαμε την `break`.

```
34          # έλεγχος αποτελέσματος
35          if newroll == roll:
36              print("Κέρδισες!")
37              # το παιχνίδι τελείωσε
38              over = True
39          elif newroll == 7:
40              print("Έχασες...")
41              # το παιχνίδι τελείωσε
42              over = True
```

craps/src/craps.4.py



Σχήμα 2.6: Όταν η `over` πάρει τιμή `True`, η επανάληψη *δεν* θα διακοπεί άμεσα, γιατί η συνθήκη `not over` της `while` δεν ελέγχεται συνεχώς. Η επανάληψη θα διακοπεί όταν ελεγχθεί η συνθήκη της `while`, δηλαδή αφού τελειώσει ο κύκλος της επανάληψης.

Στην εκδοχή αυτή η επανάληψη *δεν* θα σταματήσει άμεσα όταν η `over` γίνει αληθής, αλλά όταν ελεγχθεί η συνθήκη συνέχειας και διαπιστωθεί ότι η `over` είναι αληθής.

Μεταβλητές όπως η `over` που παίρνουν μόνο δύο τιμές και χρησιμοποιούνται ως ένδειξη ότι κάποιο γεγονός έχει συμβεί ή όχι ονομάζονται συχνά *σημαίες* (flags).

Κάν'το Μου Λιανά

Το πρόγραμμα λειτουργεί, αλλά δεν μου αρέσει ο τρόπος που είναι γραμμένο. Οι εντολές για την ρίψη μιας ζαριάς επαναλαμβάνονται αυτούσιες σε δύο σημεία του προγράμματος.

Οι ίδιες εντολές που προσομοιώνουν τη ρίψη των ζαριών χρησιμοποιούνται τόσο στην αρχική ζαριά όσο και στις επαναλαμβανόμενες ζαριές. Αυτές οι εντολές δεν είναι ούτε αναγκαίο, ούτε σκόπιμο να ξαναγράφονται. Ουσιαστικά, αποτελούν ένα ενιαίο σύνολο και υλοποιούν μια συγκεκριμένη λειτουργία. Μπορούμε λοιπόν να τις τοποθετήσουμε μέσα σ' ένα *υποπρόγραμμα*, δημιουργώντας ένα κλειστό "εξάρτημα" που υλοποιεί τη λειτουργία της ρίψης των ζαριών.

Τα υποπρογράμματα στην Python και σε πολλές άλλες γλώσσες έχουν τη μορφή *συναρτήσεων*. Έχουν ένα όνομα και μια (προαιρετική) σειρά *παραμέτρων*, εντός παρενθέσεων. Όταν κληθούν, οι συναρτήσεις δέχονται τιμές για τις παραμέτρους, τις *επεξεργάζονται* και *επιστρέφουν* ένα αποτέλεσμα.

Έχουμε ήδη χρησιμοποιήσει συναρτήσεις που παρέχει έτοιμες η Python, όπως η `print()` και η `input()`, αλλά και συναρτήσεις που ανήκουν σε βιβλιοθήκες, όπως `time.sleep()` και η `random.randint()`. Τα προγράμματά μας καλούν αυτές τις συναρτήσεις κι αυτές εκτελούνται, παρέχοντας συγκεκριμένες λειτουργίες.

Το υποπρόγραμμα `rollDice()` που ακολουθεί δεν δέχεται *παραμέτρους* (γιατί δεν χρειάζεται εξωτερικές τιμές για να λειτουργήσει), και *επιστρέφει* το άθροισμα των ενδείξεων των δύο ζαριών.

```
def rollDice():
    """ Ρίχνει δύο ζάρια και επιστρέφει
    το άθροισμα των ενδείξεών τους.
    """
    # προτροπή για ρίψη ζαριών
    print("Ρίξε τα ζάρια με ENTER...", end="")
    input()
    # τυχαίες τιμές για τα δύο ζάρια
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    # υπολογισμός και εμφάνιση ζαριάς
    roll = dice1 + dice2
    print("Έριξες", dice1, dice2, "=", roll)
    return roll
```

Σημειώστε ότι οι μεταβλητές `dice1`, `dice2` και `roll` είναι *τοπικές* στο υποπρόγραμμα. Δημιουργούνται κάθε φορά που αυτό καλείται και παύουν να υπάρχουν όταν ολοκληρωθεί η εκτέλεσή του. Η μεταβλητή `roll` του κύριου προγράμματος είναι διαφορετική από εκείνη του υποπρογράμματος.

Προς το παρόν έχουμε μόνο *ορίσει* το υποπρόγραμμα. Για να το χρησιμοποιήσουμε, ενεργοποιώντας την εκτέλεση των εντολών του, πρέπει να το *καλέσουμε* στα δύο σημεία του προγράμματος όπου ο χρήστης ρίχνει τα ζάρια, *αντικαθιστώντας* τις υπάρχουσες εντολές.

Έτσι, στο σημείο όπου ρίχνεται η αρχική ζαριά, τώρα καλούμε τη `rollDice()`, ονομάζοντας `roll` το αποτέλεσμα που επιστρέφεται.

```
# ρίψη αρχικής ζαριάς
roll = rollDice()
# έλεγχος αποτελέσματος
if roll == 7 or roll == 11:
```

Παρομοίως, στο σημείο όπου ρίχνεται η επαναλαμβανόμενη ζαριά, καλούμε και πάλι τη `rollDice()`, ονομάζοντας `newroll` το αποτέλεσμα που επιστρέφεται.

```
    # επανάληψη ρίψεων
    while not over:
        # ρίψη ζαριάς (επαναληπτική)
        newroll = rollDice()
        # έλεγχος αποτελέσματος
        if newroll == roll:
```

craps/src/craps.5.py

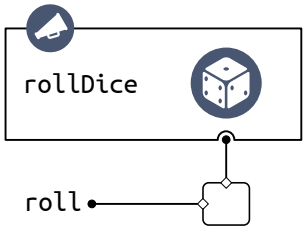
Σχήμα 2.7: Η συνάρτηση `rollDice` είναι ένα ανεξάρτητο τμήμα κώδικα, ένα αυτόνομο "εξάρτημα" που επιτελεί μια συγκεκριμένη λειτουργία. Όταν κληθεί, η συνάρτηση προσομοιώνει τη ρίψη δύο ζαριών κι επιστρέφει το άθροισμα των ενδείξεών τους.

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

Οι εντολές μετά την πρώτη γραμμή είναι στοιχισμένες δεξιότερα. Η στοίχιση αυτή υποδηλώνει ότι οι εντολές αυτές "ανήκουν" στη συνάρτηση και θα εκτελεστούν όταν αυτή *κληθεί*.

Το αποτέλεσμα της συνάρτησης (όταν υπάρχει) *επιστρέφεται* με την **`return`**.

Το ειδικό σχόλιο στην αρχή της συνάρτησης, ανάμεσα στα 3 διπλά εισαγωγικά, χρησιμοποιείται για να περιγράψει τη λειτουργία της. Είναι προαιρετικό, όπως όλα τα σχόλια.



Σχήμα 2.8: Όταν κληθεί η `rollDice()` εκτελούνται οι εντολές της και η συνάρτηση επιστρέφει μια τιμή. Για την αρχική ζαριά, η τιμή αυτή αποδίδεται στη μεταβλητή `roll` του κύριου προγράμματος.

Πλήρες Πρόγραμμα

```python
import random

def rollDice():
    """ Ρίχνει δύο ζάρια και επιστρέφει
    το άθροισμα των ενδείξεών τους.
    """
    # προτροπή για ρίψη ζαριών
    print("Ρίξε τα ζάρια με ENTER...", end="")
    input()
    # τυχαίες τιμές για τα δύο ζάρια
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    # υπολογισμός και εμφάνιση ζαριάς
    roll = dice1 + dice2
    print("Έριξες", dice1, dice2, "=", roll)
    return roll

# ρίψη αρχικής ζαριάς
roll = rollDice()
# έλεγχος αποτελέσματος
if roll == 7 or roll == 11:
    # νίκη με την πρώτη
    print("Κέρδισες με την πρώτη!")
elif roll <= 3 or roll == 12:
    # ήττα με την πρώτη
    print("Έχασες με την πρώτη...")
else:
    # τίθεται ο "στόχος"
    print("Ξαναρίξε. Πρέπει να φέρεις", roll)
    # το παιχνίδι δεν έχει τελειώσει
    over = False
    # επανάληψη ρίψεων
    while not over:
        # ρίψη ζαριάς (επαναληπτική)
        newroll = rollDice()
        # έλεγχος αποτελέσματος
        if newroll == roll:
            print("Κέρδισες!")
            # το παιχνίδι τελείωσε
            over = True
        elif newroll == 7:
            print("Έχασες...")
            # το παιχνίδι τελείωσε
            over = True
```

craps/src/craps.final.py

Ασκήσεις

2.1 *Ο Ευκλείδης, στο έβδομο βιβλίο των* Στοιχείων *του, περιγράφει μια μέθοδο για την εύρεση του μέγιστου κοινού διαιρέτη δύο ακεραίων. Τα Στοιχεία γράφτηκαν περίπου το 300 π.Χ. και η μέθοδος καθεαυτή θεωρείται ακόμα παλιότερη, γι' αυτό και ο Donald Knuth, στο εμβληματικό βιβλίο του* The Art of Computer Programming*, τη χαρακτηρίζει ως τον "παππού" όλων των αλγορίθμων, γιατί είναι ο παλαιότερος γνωστός αλγόριθμος που χρησιμοποιείται ακόμα και σήμερα.*

Μπορεί κανείς να βρει πολλές διαφορετικές εκδοχές του αλγορίθμου, αλλά βασικά ο αλγόριθμος μπορεί να συνοψιστεί ως εξής:

> *Αφαίρεσε τον μικρότερο αριθμό από τον μεγαλύτερο και επανάλαβε τη διαδικασία μέχρι οι δύο αριθμοί να γίνουν ίσοι. Η τιμή των δύο αριθμών όταν τελειώσει η διαδικασία είναι ο μέγιστος κοινός διαιρέτης τους.*

Γράψτε ένα πρόγραμμα το οποίο ζητάει από το χρήστη δύο ακέραιους αριθμούς και υπολογίζει τον μέγιστο κοινό διαιρέτη τους, χρησιμοποιώντας τη μέθοδο του Ευκλείδη.

`craps/exercises/eukleides.py`

2.2 *Το 1937, ο γερμανός μαθηματικός Lothar Collatz διατύπωσε τον παρακάτω ισχυρισμό, ο οποίος παραμένει αναπόδεικτος:*

> *Επιλέξτε έναν θετικό ακέραιο n. Αν είναι άρτιος διαιρέστε τον με το 2, ενώ αν είναι περιττός πολλαπλασιάστε τον με το 3 και προσθέστε μια μονάδα. Επαναλάβετε τη διαδικασία με τον νέο αριθμό που θα προκύψει. Από οποιονδήποτε αριθμό n κι αν ξεκινήσετε, θα καταλήξετε στον αριθμό 1.*

Η εικασία έχει επαληθευτεί αριθμητικά μέχρι και για αριθμούς της τάξης των 6 δισεκατομμυρίων δισεκατομμυρίων, ωστόσο δεν υπάρχει αναλυτική μαθηματική απόδειξη. Θεωρητικά υπάρχει πάντα το ενδεχόμενο ένας ακόμα μεγαλύτερος αριθμός να παραβιάζει την εικασία!

`xkcd.com/710/`

Γράψτε ένα πρόγραμμα το οποίο θα διαβάζει από το χρήστη τον αριθμό εκκίνησης n, θα επαναλαμβάνει τη διαδικασία που περιγράφεται παραπάνω και θα εμφανίζει τους διαδοχικούς αριθμούς που προκύπτουν από αυτή, έως ότου η διαδικασία καταλήξει στον αριθμό 1. Μπορείτε να εμπλουτίσετε το πρόγραμμά σας, ώστε μετά το τέλος της διαδικασίας να εμφανίζει το χρόνο τερματισμού, δηλαδή το συνολικό αριθμό των βημάτων που απαιτήθηκαν, αλλά και το σημείο πλημμυρίδας, δηλαδή τον μεγαλύτερο αριθμό που προέκυψε κατά την εκτέλεση της διαδικασίας.

Για να διαπιστώσετε αν ένας αριθμός είναι περιττός ελέγξτε αν το υπόλοιπο της διαίρεσής του με το 2 είναι το 1.

Για παράδειγμα, ξεκινώντας από το n=6 δημιουργείται η παρακάτω ακολουθία, με χρόνο τερματισμού τα 8 βήματα και σημείο πλημμυρίδας το 16:

6 3 10 5 16 8 4 2 1

`craps/exercises/collatz.py`

2.3 *Γράψτε μια συνάρτηση* `flip()`*, η οποία θα μιμείται τη ρίψη ενός νομίσματος, επιστρέφοντας με τυχαίο τρόπο είτε το* `0`*, είτε το* `1`*.*

Κατασκευάστε πρόγραμμα το οποίο θα παίζει Κορώνα-Γράμματα *με το χρήστη. Το πρόγραμμα θα ρωτάει τον παίκτη αν επιλέγει Κορώνα ή Γράμματα (*`0` *για Κορώνα και* `1` *για Γράμματα), θα προσομοιώνει τη ρίψη του νομίσματος καλώντας τη* `flip` *και θα ενημερώνει τον παίκτη αν κέρδισε ή έχασε. Κάντε το πρόγραμμά σας επαναληπτικό, με το παιχνίδι να σταματά όταν ο χρήστης σε κάποιο γύρο δεν επιλέξει ούτε Κορώνα, ούτε Γράμματα.*

craps/exercises/headstails.py

Μπορείτε να τροποποιήσετε το πρόγραμμά σας έτσι ώστε να "κλέβει" το χρήστη. Χωρίς να γίνεται ρίψη του νομίσματος, κάντε το πρόγραμμα ν' ανακοινώνει ότι το αποτέλεσμα της ρίψης του κέρματος ήταν το αντίθετο από αυτό που επέλεξε ο παίκτης.

craps/exercises/headstails-cheat.py

Για να γίνει το πρόγραμμα πιο πειστικό, καλό θα ήταν να μην κερδίζει πάντα, αλλά ν' αφήνει μερικές φορές και το χρήστη να κερδίσει και να αισθανθεί τυχερός... Σε αυτές τις περιπτώσεις, χωρίς να γίνεται ρίψη του νομίσματος, το πρόγραμμα θ' ανακοινώνει ότι το αποτέλεσμα της ρίψης του κέρματος ήταν το ίδιο με αυτό που επέλεξε ο παίκτης.

craps/exercises/headstails-cheat-probability.py

Υπόδειξη: Πριν την ανακοίνωση του "στημένου" αποτελέσματος, το πρόγραμμα μπορεί να επιλέγει τυχαία έναν αριθμό από το `1` μέχρι το `100`. Αν αυτός ο αριθμός ξεπερνά ένα υψηλό "κατώφλι", τότε το πρόγραμμα θ' ανακοινώνει στον παίκτη ότι κέρδισε.

```
win = random.randint(1,100)
if win > 90:
    # ο παίκτης κερδίζει
    ...
else:
    # ο παίκτης χάνει
    ...
```

2.4 *Να κατασκευάσετε ένα πρόγραμμα το οποίο θα παίζει επαναληπτικά το γνωστό παιχνίδι «Πέτρα-Ψαλίδι-Χαρτί» με το χρήστη. Σε κάθε γύρο, το πρόγραμμα θα ζητά από το χρήστη την επιλογή του (*`1` *για την Πέτρα,* `2` *για το Ψαλίδι ή* `3` *για το Χαρτί) και στη συνέχεια θα εμφανίζει τη δική του επιλογή και θα ανακοινώνει το νικητή του γύρου. Αν ο χρήστης πληκτρολογήσει οτιδήποτε εκτός από τις τρεις έγκυρες επιλογές τότε θεωρείται ότι επιθυμεί να τερματίσει το παιχνίδι.*

craps/exercises/rps.py

Μπορείτε να δοκιμάσετε και παραλλαγές του παιχνιδιού: τροποποιήστε το πρόγραμμα έτσι ώστε να επιλέγει από τις διαφορετικές κινήσεις με διαφορετική πιθανότητα ή κάντε το πρόγραμμα να κλέβει (μερικές φορές), επιλέγοντας κίνηση με βάση την κίνηση του χρήστη.

craps/exercises/rps-cheat.py

2.5 *Το «Ανάμεσα» ή «Acey Ducey» είναι ένα παιχνίδι με χαρτιά. Ο ένας παίκτης (η «μάνα») τραβάει δύο χαρτιά από την τράπουλα και τα δείχνει στο δεύτερο παίκτη. Εκείνος με την σειρά του στοιχηματίζει ένα ποσό*

και νικάει όταν το τρίτο χαρτί είναι ανάμεσα στα δύο πρώτα. Όσο πιο απίθανο είναι να νικήσει ένας παίκτης, τόσο μεγαλύτερο είναι το ποσό που θα κερδίσει. Συγκεκριμένα, αν υπάρχουν d χαρτιά ανάμεσα στα δύο πρώτα, τότε το ποσό που κερδίζει ο παίκτης σε περίπτωση νίκης θα είναι 12/d φορές το ποσό που στοιχημάτισε.

Για παράδειγμα, αν τα δύο πρώτα χαρτιά είναι το 5 και το 8, τότε ο δεύτερος παίκτης θα νικήσει αν το επόμενο χαρτί είναι το 6 ή το 7 (d=2). Στην περίπτωση αυτή θα κερδίσει 6 φορές το ποσό που στοιχημάτισε.

Θεωρήστε ότι τα χαρτιά που χρησιμοποιούνται είναι από το 1 μέχρι και το 13 (τα τρία τελευταία αντιστοιχούν στον βαλέ, τη ντάμα και τον ρήγα). Αναπτύξτε πρόγραμμα το οποίο παράγει τυχαία τα δύο πρώτα χαρτιά της μάνας. Αυτά θα πρέπει να διαφέρουν αριθμητικά τουλάχιστον κατά δύο μονάδες, ώστε να υπάρχει περιθώριο για τουλάχιστον ένα χαρτί ανάμεσά τους ($d \geq 1$), αλλιώς η μάνα θα πρέπει να μοιράσει δύο νέα χαρτιά. Στη συνέχεια, το πρόγραμμα ρωτά τον παίκτη το ποσό που επιθυμεί να στοιχηματίσει και παράγει το τρίτο χαρτί, εμφανίζοντας κατάλληλο μήνυμα με το ποσό που κέρδισε ή έχασε ο δεύτερος παίκτης.

`craps/exercises/aceyducey.py`

Δομη Επαναληψης *Η δομή επανάληψης μας δίνει τη δυνατότητα να περιγράφουμε διαδικασίες που επαναλαμβάνονται. Ίσως αρχικά αυτό να μην ακούγεται ιδιαίτερα εντυπωσιακό, γρήγορα όμως ανακαλύπτουμε ότι οι περισσότερες υπολογιστικές διαδικασίες είναι εγγενώς επαναληπτικές (αυτό περιλαμβάνει και τα περισσότερα* παιχνίδια*) και είναι απαραίτητος ένας συμπαγής τρόπος περιγραφής τους. Με τη δομή επανάληψης περιγράφουμε τα βήματα ενός μόνο κύκλου, ακόμα κι αν τελικά αυτά τα βήματα θα εκτελεστούν πάρα πολλές φορές. Ή ακόμα κι αν δεν γνωρίζουμε εκ των προτέρων πόσες φορές θα εκτελεστούν. Από μία άποψη, στις υπολογιστικές μας συσκευές ταιριάζει πολύ η δομή επανάληψης: σε αντίθεση με τους ανθρώπους, έχουν τη δυνατότητα να εκτελούν αδιαμαρτύρητα τα ίδια βήματα, ξανά και ξανά.*

Αναγνωσιμοτητα και Break *Σε μια επανάληψη μπορούμε να χρησιμοποιήσουμε μία ή και περισσότερες* **break**. *Κάθε* **break** *ανοίγει μια πόρτα εξόδου από τον κύκλο της επανάληψης. Επομένως, για να γνωρίζει κανείς πότε θα τερματιστεί μια επανάληψη θα πρέπει να αναζητήσει τις* **break** *μέσα στην επανάληψη και να διαπιστώσει υπό ποιες συνθήκες θα εκτελεστούν. Αντίθετα, χωρίς την* **break**, *η έξοδος από την επανάληψη γίνεται από ένα και μοναδικό σημείο: την συνθήκη συνέχειας. Η επανάληψη τερματίζεται μόνο όταν η συνθήκη συνέχειας ελεγχθεί και διαπιστωθεί ότι είναι ψευδής. Αρκεί λοιπόν να κοιτάξει κανείς την συνθήκη συνέχειας για να γνωρίζει πότε θα τερματιστεί μια επανάληψη. Φαίνεται λοιπόν ότι η άμεση διακοπή μιας επανάληψης με την* **break** *είναι μεν συχνά βολική, όμως χωρίς την* **break** *προκύπτουν προγράμματα που είναι περισσότερο κατανοητά και ευανάγνωστα. Αυτό είναι εξαιρετικά σημαντικό για κάποιους και πολύ λιγότερο για άλλους...*

Υποπρογραμματα *Το μεγαλύτερο πλεονέκτημα που προκύπτει από τη χρήση υποπρογραμμάτων (και γίνεται πολύ εμφανέστερο καθώς τα προβλήματα και τα αντίστοιχα προγράμματα μεγαλώνουν) είναι ότι είμαστε "αναγκασμένοι" να αναλύουμε τα προβλήματα και να τ' αντιμετωπίζουμε* τμηματικά. *Κάθε φορά εστιάζουμε την προσοχή μας σε μικρότερα κι απλούστερα κομμάτια του γενικού προβλήματος και στη συνέχεια, συνθέτουμε τη λύση, συναρμολογώντας τα κομμάτια αυτά. Στα κεφάλαια που ακολουθούν θα χρησιμοποιήσουμε περισσότερο τα υποπρογράμματα και θα αναφερθούμε εκτενέστερα στα πλεονεκτήματα που πηγάζουν από τη χρήση τους.*

Τυχαιοι Αριθμοι *Η παραγωγή τυχαίων αριθμών έχει πολλές και σημαντικές εφαρμογές, όπως στην στατιστική, τις προσομοιώσεις και τα παιχνίδια, όμως η σημαντικότερη εφαρμογή της είναι στην* κρυπτογραφία. *Υπάρχουν αρκετές υπολογιστικές μέθοδοι για να παράγει κανείς τυχαίους αριθμούς. Όλες τους ονομάζονται και* γεννήτριες

ψευδο-τυχαίων αριθμών *γιατί η αλήθεια είναι ότι οι αριθμοί που παράγουν* φαίνονται *τυχαίοι, αλλά δεν είναι. Υπάρχουν μάλιστα και ειδικά στατιστικά κριτήρια που μετρούν πόσο απρόβλεπτες είναι οι ακολουθίες αριθμών που παράγονται. Η Python (και άλλες γλώσσες) χρησιμοποιούν έναν αλγόριθμο που ονομάζεται Mersenne-Twister, ο οποίος είναι επαρκής για τις συνήθεις χρήσεις αλλά όχι για την κρυπτογραφία.*

Ζαρια, Χαρτια και Τυχερα Παιχνιδια *Πολλά από τα παραδείγματα με τα οποία θ' ασχοληθούμε είναι παιχνίδια με ζάρια ή χαρτιά. Τα επιλέξαμε επειδή θεωρούμε ότι είναι διασκεδαστικά και έχουν αλγοριθμικό ενδιαφέρον. Ωστόσο, ο* εθισμός *σε αυτά τα παιχνίδια αποτελεί σημαντικότατο πρόβλημα. Τα* ηλεκτρονικά *τυχερά παιχνίδια έχουν κι άλλες υπόγειες διαστάσεις. Μέσα από ορισμένες ασκήσεις, ελπίζουμε να κατανοήσετε πως όταν παίζεις ένα τυχερό παιχνίδι μ' ένα πρόγραμμα, ίσως το παιχνίδι να μην αφήνει και τόσα πολλά στην τύχη…*