Python
Συναρτησεις, Λιστες & Συμβολοσειρες – Αξιοποιωντας Δομες Δεδομενων

Προγραμματισμού και Μεταφραστών

Γιάννης Γαροφαλάκης - Σπύρος Σιούτας - Γιάννης Τζήμας

- Σε αυτό το μάθημα θα :
  - Δούμε από πιο κοντά τις συναρτήσεις, λίστες και συμβολοσειρές της Python και,
  - Θα αξιοποιήσουμε δομές δεδομένων με στόχο την επίτευξη ενός βαθμού αφαίρεσης στον προγραμματισμό.

Ας αρχίσουμε φτιάχνοντας ένα παιχνίδι με την Python

- Στο παιχνίδι αυτό ο χρήστης θα πρέπει να μαντέψει μία λέξη επιλέγοντας γράμματα.
- Θα φτιάξουμε τη γνωστή σε όλους κρεμάλα.
- Για να φτιάξουμε ένα τέτοιο πρόγραμμα με Python, θα πρέπει να μάθουμε να χρησιμοποιούμε συναρτήσεις (functions), καθώς και λίστες (lists) και συμβολοσειρές (strings).
- Και τα τρία αυτά είναι αφαιρέσεις (abstractions) που μας βοηθάνε να κάνουμε τα προγράμματά μας πιο απλά.

Ξεκινάμε με Σχεδίαση από Επάνω προς τα Κάτω – Top-down design

```
print("Game started")
```

Ενημερώνουμε το χρήστη ότι το παιχνίδι ξεκίνησε.

```
print("Game started")
playKremalaGame()
```

Καλούμε μία συνάρτηση που θα εκτελεί ολόκληρη τη βασική λογική του παιχνιδιού, κρύβοντας την πολυπλοκότητα του προγράμματος. Θα μπορούσε αργότερα να αποτελεί επιλογή ενός μενού παιχνιδιών.

```
def playKremalaGame():


print("Game started")
playKremalaGame()
```

Ορίζουμε τη συνάρτηση στην αρχή το προγράμματος με τη **`def`**, ενώ προσέχουμε να υπάρχουν οι παρενθέσεις και ο χαρακτήρας «**:**» αμέσως μετά.

```
def playKremalaGame():
    print("playing")

print("Game started")
playKremalaGame()
```

Προσθέτουμε λειτουργικότητα εντός της συνάρτησης βάζοντας εντολές. Οι εντολές θα πρέπει να βρίσκονται σε εσοχή κάτω από τον ορισμό.
Προσοχή το πρόγραμμα θα ξεκινήσει να εκτελείται από την **`print("Game started")`** και όχι από την πρώτη γραμμή, καθώς εκεί είναι ο ορισμός της συνάρτησης.

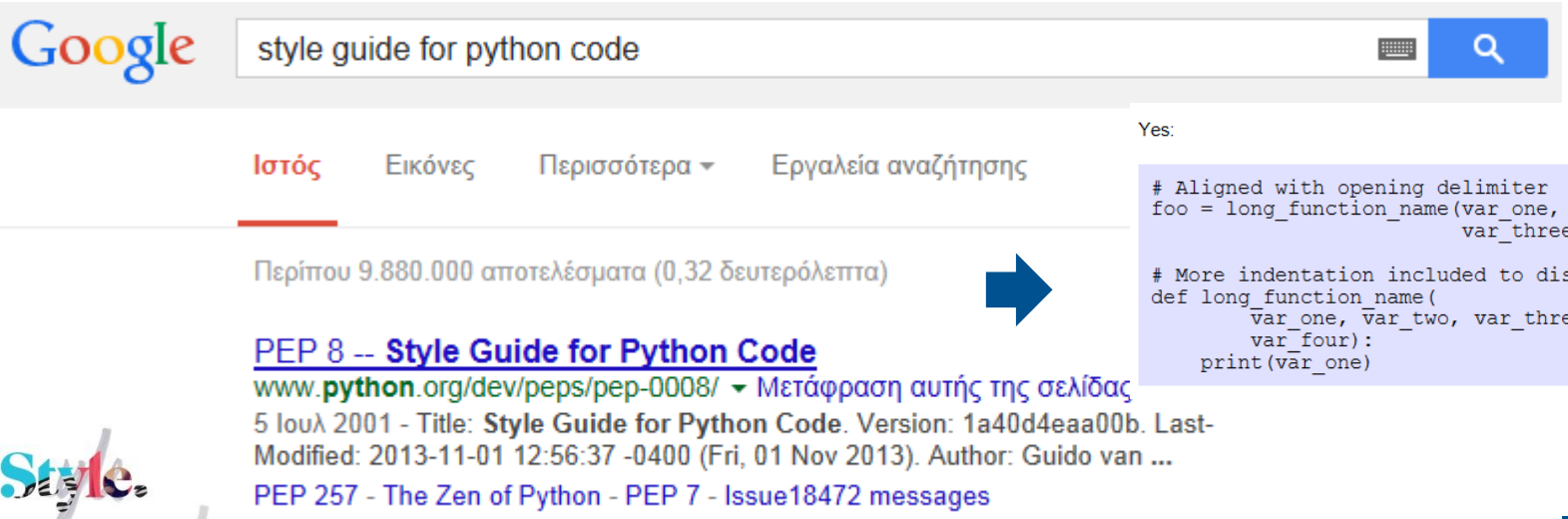
```
def playKremalaGame():
    print("playing")

print("Game started")
playKremalaGame()
print("Game over")
```

```
>>> ================================ RESTART ================================
>>>
Game started
playing
Game over
>>>
```

Προγραμματίζοντας με στυλ

- Κάθε γλώσσα προγραμματισμού έχει το δικό της στυλ γραψίματος.
- Όταν λέμε στυλ εννοούμε συμβάσεις ανάμεσα στους προγραμματιστές για τον τρόπο που γίνεται η ονοματοδοσία των μεταβλητών, πως μπαίνουν κενά κ.α..
- Δεν είναι απαραίτητο να το ακολουθήσετε, αλλά καλό θα ήταν να το κάνετε αν πρόκειται να δώσετε τον κώδικά σας σε άλλους προγραμματιστές.
- Το σημαντικό είναι να είστε συνεπείς στο στυλ που θα επιλέξετε!

```
def play_kremala_game():
    print("playing")

print("Game started")
play_kremala_game()
print("Game over")
```



Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Παραγωγή Τυχαίων Λέξεων Α' Επίπεδο

- Χρησιμοποιώντας top-down σχεδιασμό, τώρα μπορούμε να κάνουμε το ίδιο ακριβώς και στη συνάρτηση που κατασκευάσαμε πριν λίγο – `play_kremala_game()`.
- Δηλαδή, να μπούμε σε επόμενο επίπεδο λεπτομέρειας, και σιγά σιγά να σπάσουμε το πρόβλημα σε μικρότερα κομμάτια, χωρίς όμως να φτάσουμε στην τελική λύση.

```
def play_kremala_game():
    print("playing")

print("Game started")
play_kremala_game()
print("Game over")
```

```
def get_random_word():
    return "pizza"

def play_kremala_game():
    word = get_random_word()

print("Game started")
play_kremala_game()
print("Game over")
```

Θα πρέπει να υλοποιήσω τη συνάρτηση σε επόμενο επίπεδο, αλλά προς το παρόν ας κάνει κάτι απλό, όπως το να επιστρέφει μία λέξη.

Θα πρέπει σίγουρα να παράγω μία τυχαία λέξη την οποία ο χρήστης θα πρέπει να μαντέψει. Άρα θα πρέπει να υποθέσω ότι θα υπάρχει μία συνάρτηση για να κάνει αυτή τη δουλειά και να κάνω και τον αντίστοιχο ορισμό. Την τιμή που θα επιστρέφει αυτή η συνάρτηση την αναθέτω σε μία μεταβλητή – `word`.

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Πόσες προσπάθειες έχει ο χρήστης

```
def get_random_word():
    return "pizza"

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()

    while playing:
        strikes += 1

        if strikes >= max_strikes:
            playing = False

    if strikes >= max_strikes:
        print("Loser!")
    else:
        print("Winner!")

print("Game started")
play_kremala_game()
print("Game over")
```

Σίγουρα χρειάζομαι μία μεταβλητή που θα αποθηκεύει τον αριθμό των προσπαθειών που έκανε ο χρήστης.

Χρειάζομαι επίσης μία μεταβλητή που θα αποθηκεύει το μέγιστο αριθμό προσπαθειών του χρήστη

Χρειαζόμαστε σίγουρα ένα βρόχο ώστε να ελέγχουμε των αριθμό των προσπαθειών του χρήστη.

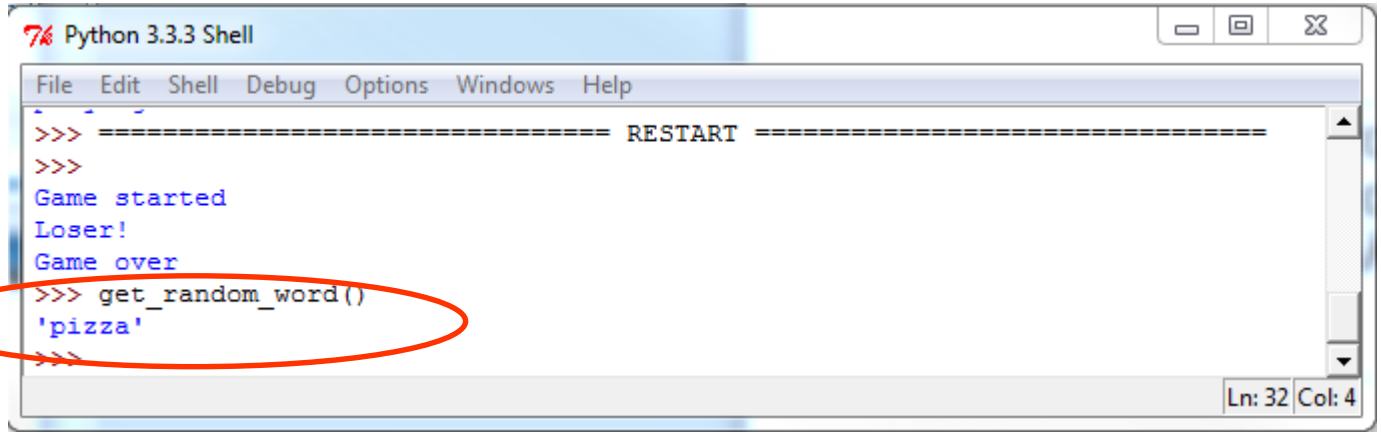
Τέλος, χρειάζεται να φτιάξουμε ένα τμήμα κώδικα που να ενημερώνει το χρήστη για το αν κέρδισε ή έχασε.

Προφανώς, όπως είναι γραμμένος ο κώδικας μέχρι στιγμής πάντα ο χρήστης θα χάνει, αλλά έχουμε προχωρήσει σε επόμενο επίπεδο λεπτομέρειας.

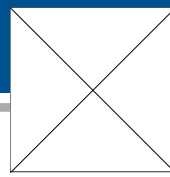
```
>>> ================================ RESTART ================================
>>>
Game started
Loser!
Game over
>>>
```

Και κάτι ενδιαφέρον

- Από τη στιγμή που έχουμε τρέξει το πρόγραμμα μπορούμε να συνεχίσουμε να αλληλεπιδρούμε με τις συναρτήσεις του στο Python Shell, αλλά αυτό δε συμβαίνει πάντα.



Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Παραγωγή μιας τυχαίας λέξης – Λίστες (1/3)



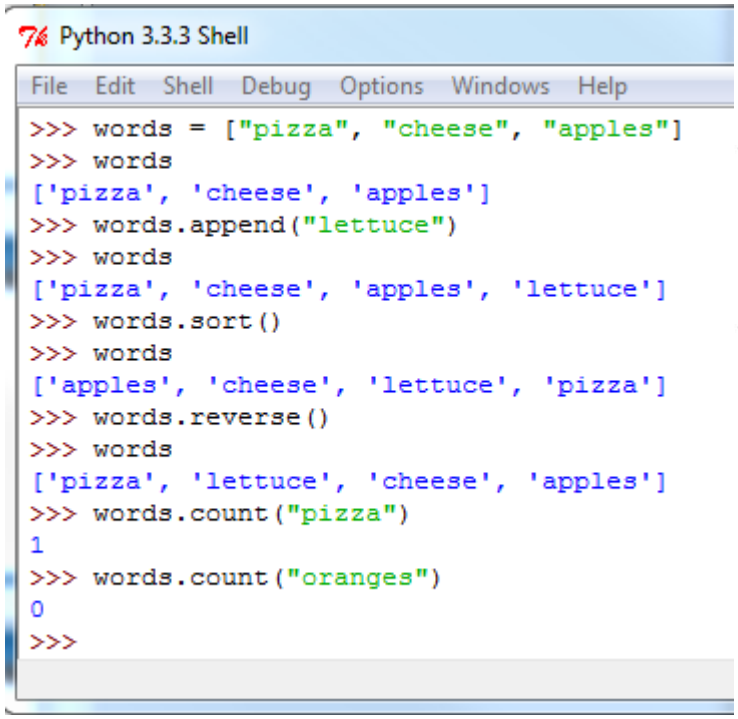
- Για να επιλέξουμε μία τυχαία λέξη, αυτό που θέλουμε από το πρόγραμμα είναι να διαθέτει μία λίστα από λέξεις και να επιλέγει μία τυχαία από αυτές. Και όταν λέμε λίστα, εννοούμε ένα τύπο αντικειμένου στην Python που είναι λίστα (list).

- Είναι στην πραγματικότητα αυτό που στον προγραμματισμό ονομάζουμε δομή δεδομένων (data structure). Μία δομή δεδομένων αποθηκεύει και οργανώνει δεδομένα για εμάς.

- Υπάρχουν διάφορες δομές δεδομένων που μπορείτε να αξιοποιήσετε. Κάθε μία από αυτές έχει δυνατά και αδύνατα σημεία. Κάποιες δομές είναι αποτελεσματικές στην αποθήκευση δεδομένων, ενώ άλλες μπορεί να είναι αργές σε αυτό, αλλά να επιτρέπουν γρήγορη αναζήτηση και ανάκτηση δεδομένων.

- Μία λίστα στην Python είναι μια δομή δεδομένων που μας επιτρέπει να αποθηκεύσουμε μηδέν ή περισσότερα στοιχεία (αριθμούς, συμβολοσειρές, αντικείμενα ή και τα τρία μαζί μέσα στην ίδια λίστα).
- Στην περίπτωσή μας θα αποθηκεύσουμε τις λέξεις που χρειαζόμαστε.

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Παραγωγή μιας τυχαίας λέξης – Λίστες (2/3)



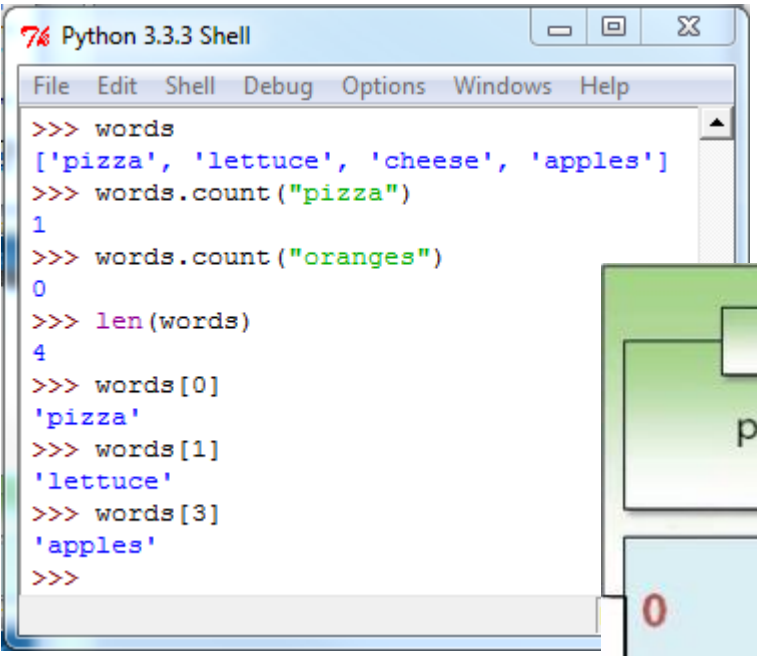
- Μπορούμε να δημιουργήσουμε μία λίστα απευθείας στο Python Shell.
- Μπορούμε να φτιάξουμε μία μεταβλητή τύπου λίστας στην οποία θα αποθηκεύσουμε τις λέξεις.
- Προσοχή, τα αντικείμενα της λίστας μπαίνουν ανάμεσα στους χαρακτήρες [και]

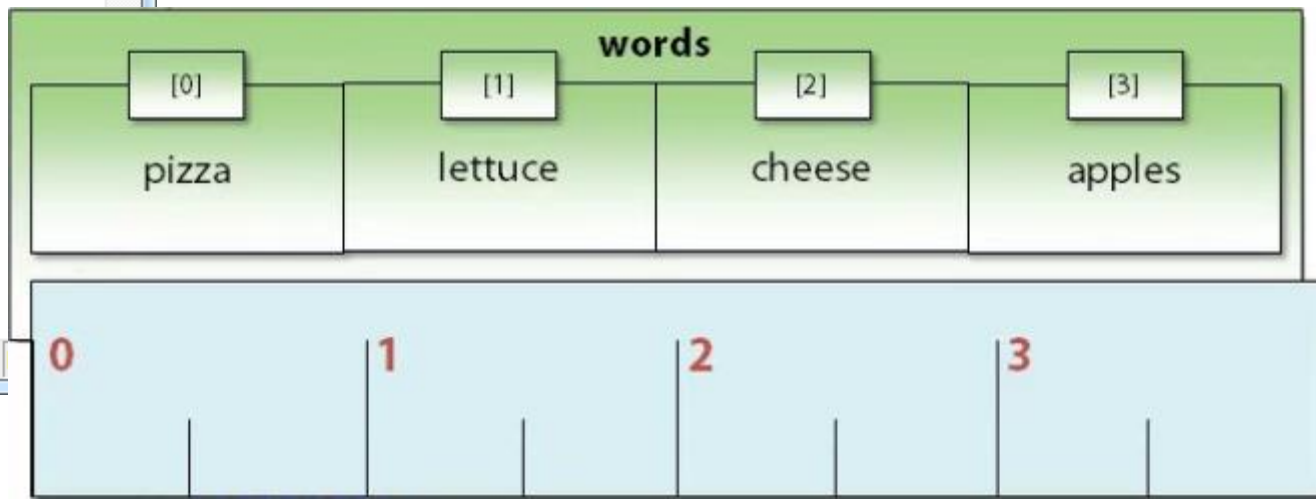
- Και από τη στιγμή που έχουμε φτιάξει τη λίστα μπορούμε να κάνουμε διάφορα ενδιαφέροντα πράγματα με αυτή, όπως το
  - να προσθέσουμε στοιχεία - `.append()`,
  - να την ταξινομήσουμε - - `.sort()`,
  - να αντιστρέψουμε τη σειρά των στοιχείων της - `. reverse()`,
  - να μετρήσουμε το πόσες φορές εμφανίζεται ένα στοιχείο - `.count()`.

- Και όλα τα παραπάνω είναι λειτουργίες που είναι μέθοδοι της ίδιας της λίστας – δηλαδή συναρτήσεις τις οποίες μπορώ να καλέσω βάζοντας τελεία μετά τη μεταβλητή.

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Παραγωγή μιας τυχαίας λέξης – Λίστες (3/3)

- Υπάρχουν όμως και άλλα ενδιαφέροντα πράγματα που μπορώ να κάνω, τα οποία δεν είναι μέθοδοι της λίστας, αλλά ενσωματωμένες συναρτήσεις της Python. Μπορώ να:
  - βρω το μήκος μιας λίστας – `len(words)`. Το ίδιο βέβαια μπορώ να κάνω και με ένα string,
  - αναφερθώ με δείκτη (index) για να ανασύρω ένα στοιχείο της λίστας – `words[0]`. Προσοχή, η πρώτη θέση της λίστας είναι η `[0]`. Το ίδιο συμβαίνει και σε άλλες γλώσσες.

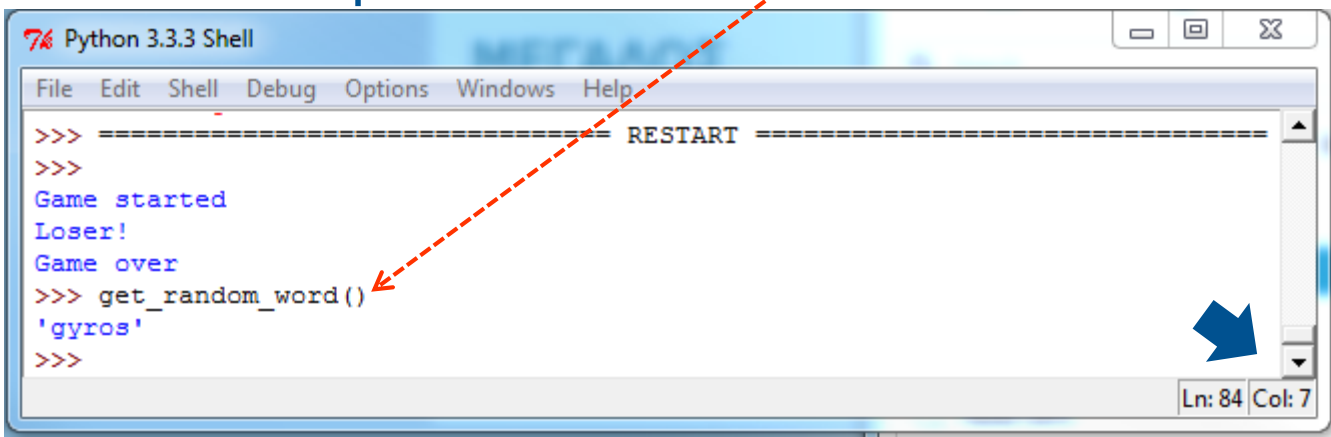




Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Παραγωγή μιας τυχαίας λέξης – Πίσω στο πρόγραμμα

```
import random

def get_random_word():
    words = ["pizza", "souvlaki", "gyros"]
    word = words[random.randint(0, len(words)-1)]
    return word

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()

    while playing:
        strikes += 1

        if strikes >= max_strikes:
            playing = False

    if strikes >= max_strikes:
        print("Loser!")
    else:
        print("Winner!")

print("Game started")
play_kremala_game()
print("Game over")
```

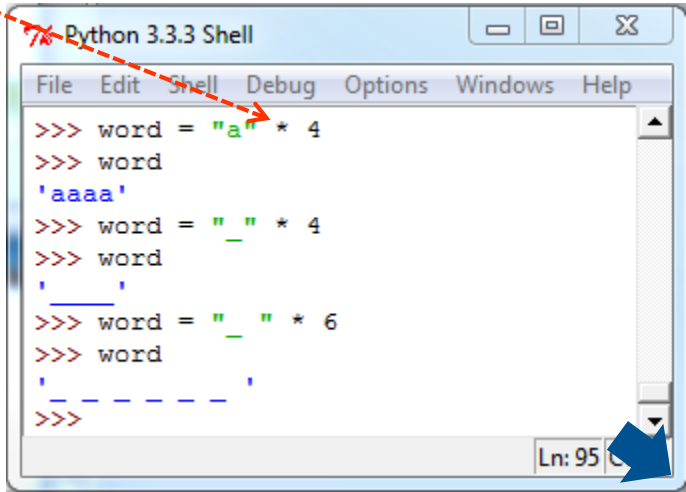
- Ορίζουμε μία μικρή λίστα με λέξεις.
- Επιλέγουμε μία λέξη από τη λίστα αξιοποιώντας τη len() και τη randint() που ήδη ξέρουμε. Προσοχή πρέπει να γίνει import πρώτα.
- Και τέλος μέσα από το Shell κάνουμε ένα πρώτο τεστ του κώδικα.



Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Ας μαντέψει ο χρήστης τα γράμματα (1/6)

```
import random

def get_random_word():
    words = ["pizza", "souvlaki", "gyros"]
    word = words[random.randint(0, len(words)-1)]
    return word

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = "_" * len(word)

    while playing:
        show_word(blanked_word)
        letter = get_guess()

        strikes += 1

        if strikes >= max_strikes:
            playing = False

    if strikes >= max_strikes:
        print("Loser!")
    else:
        print("Winner!")

print("Game started")
play_kremala_game()
print("Game over")
```

- Καλό θα ήταν πρώτα να δείξουμε οπτικά στο χρήστη πόσα γράμματα θα πρέπει να μαντέψει βάζοντας αντίστοιχες θέσεις γραμμάτων.
- Ας αξιοποιήσουμε ένα χαρακτηριστικό της Python που είναι ενδιαφέρον.
- Σκεπτόμενοι top-down, ενώ παίζουμε το πρώτο πράγμα που θα πρέπει να κάνουμε θα είναι να δείξουμε στο χρήστη την κρυμμένη λέξη και να τον αφήσουμε να μαντέψει ένα γράμμα.



Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Ας μαντέψει ο χρήστης τα γράμματα (2/6)

- Ας ξεκινήσουμε από τη `show_word()`
- Δεν θα έχουμε όμως το αποτέλεσμα που θέλουμε. Ο χρήστης δε θα είναι σε θέση να δει τον αριθμό των χαρακτήρων
- Ας παίξουμε λίγο με το Python Shell για να δούμε τι μπορούμε να κάνουμε αξιοποιώντας ένα loop για να διατρέξουμε τη λέξη:
  - Παρατηρείστε ότι εκτυπώνονται έξι κενά, με αλλαγή γραμμής.
  - Αλλά εμείς δε θέλουμε αυτό.
- Αν αξιοποιήσουμε τη σύνταξη της `print()`, μπορούμε να βγάλουμε την αλλαγή γραμμής από το τέλος κάθε χαρακτήρα και να προσθέσουμε ένα κενό μετά τον χαρακτήρα _

```
def show_word(word):
    print(word)
```

```
>>> word = "_" * 6
>>> word
'______'
>>> print(word)
______
```

```
>>> for character in word:
        print(character)


_
_
_
_
_
_
```

```
>>> for character in word:
        print(character, " ", end="")


_ _ _ _ _ _
>>>
```

Named parameter

```
print(
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Ας μαντέψει ο χρήστης τα γράμματα (3/6)

```
import random

def get_random_word():
    words = ["pizza", "souvlaki", "gyros"]
    word = words[random.randint(0, len(words)-1)]
    return word

def show_word(word):
    for character in word:
        print(character, " ", end="")
    print("")

def get_guess():
    print("Enter a letter: ")
    return input()

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = "_" * len(word)

    while playing:
        show_word(blanked_word)
        letter = get_guess()

        strikes += 1

        if strikes >= max_strikes:
            playing = False

    if strikes >= max_strikes:
```

- Αφού είμαστε έτοιμοι με τη `show_word()`, ας προχωρήσουμε με τη `get_guess()`.
- Απλά θα ζητήσουμε από το χρήστη να μας δώσει ένα γράμμα.
- Και αν το τρέξουμε:

```
>>> ================================ RESTART ================
>>>
Game started

_ _ _ _ _
Enter a letter:
a

_ _ _ _ _
Enter a letter:
b

_ _ _ _ _
Enter a letter:
a
Loser!
Game over
>>>
```

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Ας μαντέψει ο χρήστης τα γράμματα (4/6)

```
import random

def get_random_word():
    words = ["pizza", "souvlaki", "gyros"]
    word = words[random.randint(0, len(words)-1)]
    return word

def show_word(word):
    for character in word:
        print(character, " ", end="")
    print("")

def get_guess():
    print("Enter a letter: ")
    return input()

def process_letter(letter, sercret_word):
    result = False


    return result

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = "_" * len(word)

    while playing:
        show_word(blanked_word)
        letter = get_guess()
        found = process_letter(letter, word)

        strikes += 1
```

- Αφού έχουμε καταφέρει να δείχνουμε στο χρήστη τον αριθμό γραμμάτων της λέξης, ας δούμε τώρα που θα επεξεργαστούμε τα γράμματα που μαντεύει ο χρήστης.
- Σε πρώτο στάδιο θα χρειαστούμε σίγουρα μία συνάρτηση η οποία θα επεξεργάζεται την είσοδο που έδωσε ο χρήστης - `process_letter()`.
- Αυτό που θέλουμε η `process_letter()` να μας επιστρέφει είναι το αν βρήκε ο χρήστης το γράμμα ή όχι.
- Προσοχή οι μεταβλητές letter και word είναι τοπικές στην `play_kremala_game()` και για το λόγο αυτό τις περνάμε σαν παραμέτρους στην `process_letter()`.

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Ας μαντέψει ο χρήστης τα γράμματα (5/6)

```
import random

def get_random_word():
    words = ["pizza", "souvlaki", "gyros"]
    word = words[random.randint(0, len(words)-1)]
    return word

def show_word(word):
    for character in word:
        print(character, " ", end="")
    print("")

def get_guess():
    print("Enter a letter: ")
    return input()

def process_letter(letter, sercret_word):
    result = False

    for i in range(0, len(secret_word)):
        if secret_word[i] == letter:
            result = True

    return result

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = "_" * len(word)

    while playing:
        show_word(blanked_word)
        letter = get_guess()
        found = process_letter(letter, word)

        strikes += 1
```

- Βάζοντας λογική στην `process_letter()`, θα πρέπει να διαπεράσουμε τη λέξη και να δούμε αν το γράμμα υπάρχει μέσα σε αυτή.

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Ας μαντέψει ο χρήστης τα γράμματα (6/6)

```
import random

def get_random_word():
    words = ["pizza", "souvlaki", "gyros"]
    word = words[random.randint(0, len(words)-1)]
    return word

def show_word(word):
    for character in word:
        print(character, " ", end="")
    print("")

def get_guess():
    print("Enter a letter: ")
    return input()

def process_letter(letter, secret_word, blanked_word):
    result = False

    for i in range(0, len(secret_word)):
        if secret_word[i] == letter:
            result = True
            blanked_word[i] = letter

    return result

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = list("_" * len(word))

    while playing:
        show_word(blanked_word)
        letter = get_guess()
        found = process_letter(letter, word, blanked_word)

        strikes += 1
```

- Και μιας και η `process_letter()` ήδη διαπερνά τη λέξη, ίσως θα ήταν καλό να ενημερώσουμε και το αποτέλεσμα που θα επιστρέψουμε στο χρήστη αν υπάρχει επιτυχία στο γράμμα που έχει μαντέψει.
- Το μόνο που έχουμε να κάνουμε είναι να περάσουμε σαν παράμετρο στη συνάρτηση `process_letter()` και τη `blanked_word`.
- Και προσοχή τη `blanked_word` θα πρέπει να την κάνουμε λίστα.

Top-down design, πηγαίνοντας σε επόμενο επίπεδο λεπτομέρειας – Το τελικό στάδιο...

```
def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = list("_" * len(word))

    while playing:
        show_word(blanked_word)
        letter = get_guess()
        found = process_letter(letter, word, b

        strikes += 1

        if strikes >= max_strikes:
            playing = False

    if strikes >= max_strikes:
        print("Loser!")
    else:
        print("Winner!")
```

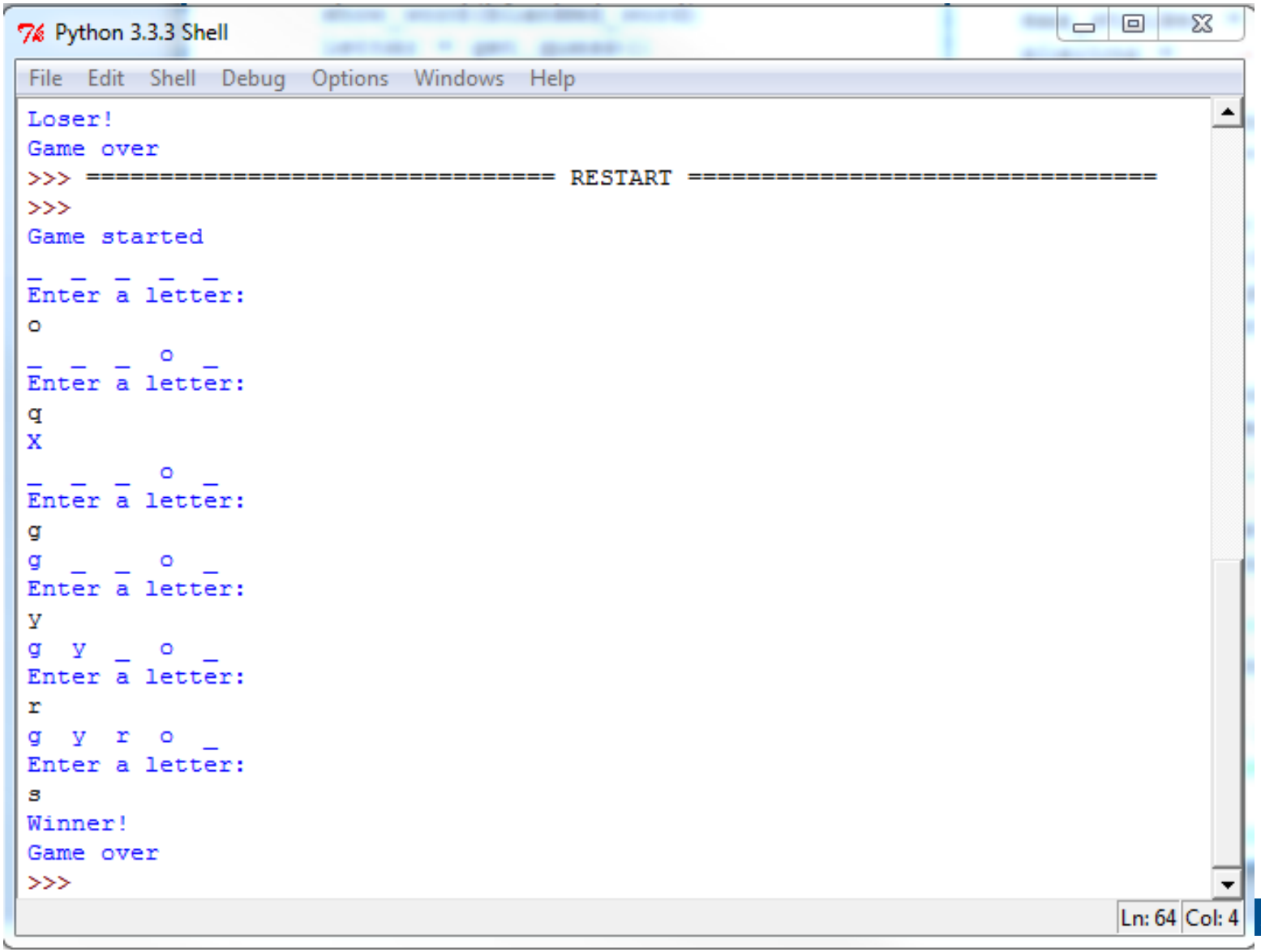
- Το μόνο που έχουμε να κάνουμε είναι να βάλουμε τη λογική αποφάσεων για το αν ο χρήστης κέρδισε ή έχασε στην `play_Kremala_game()`.

```
def print_strikes(number_of_strikes):
    for i in range(0, number_of_strikes):
        print("X", end="")
    print("")

def play_kremala_game():
    strikes = 0
    max_strikes = 3
    playing = True

    word = get_random_word()
    blanked_word = list("_" * len(word))

    while playing:
        show_word(blanked_word)
        letter = get_guess()
        found = process_letter(letter, word, blanked_word)

        if not found:
            strikes += 1
            print_strikes(strikes)

        if strikes >= max_strikes:
            playing = False

        if not "_" in blanked_word:
            playing = False

    if strikes >= max_strikes:
        print("Loser!")
    else:
        print("Winner!")
```

Εκτυπώνουμε τον αριθμό των φορών που έχουμε χάσει

Απλά ελέγχουμε αν υπάρχουν κενά στη λέξη

Και το αποτέλεσμα…



Τι είδαμε μέχρι τώρα;

- Φτιάξαμε ένα παιχνίδι με Python αξιοποιώντας top-down design.
- Η προσέγγιση αυτή, δηλαδή το να σπάμε ένα πρόγραμμα σε μικρότερα κομμάτια, οδηγεί σε προγράμματα που είναι:
  - Πιο εύκολα στο να τα κατανοήσουμε.
  - Πιο εύκολα στο να τα αλλάξουμε.
  - Και πιο ευέλικτα.
- Η αφαίρεση έχει σχέση με όλα τα παραπάνω. Γενικεύουμε τις λεπτομέρειες ώστε να μην ασχολούμαστε με αυτές μέχρι τη στιγμή που πρέπει.

- …φτιάχνοντας ένα καλάθι αγορών.
- Ο χρήστης θα μπορεί να προσθέσει τα αντικείμενα που θέλει να αγοράσει και θα συντηρούμε μία δομή δεδομένων που θα διατηρεί όλα τα αντικείμενα που θέλει να αγοράσει ο χρήστης.
- Κάνοντας όλα αυτά θα ξαναδούμε τις λίστες (lists), αλλά θα δούμε και πλειάδες (tuples), σύνολα (sets) και λεξικά (dictionaries).

Ξεκινάμε πάλι με την ίδια φιλοσοφία...

```
def get_item():
    print("Please enter an item: ")
    return input()


def go_shopping():
    cart = []

    while True:
        item = get_item()
        if item == "":
            break
        cart.append(item)

    print(cart)
    print("Finished!")

go_shopping()
```

1. Ξεκινάμε τα ψώνια μας.

2. Θα πρέπει να φτιάξουμε το καλάθι μας με μία κενή λίστα.

3. Θα πρέπει να φτιάξουμε ένα loop για να πάρουμε τα αντικείμενα που θέλει ο χρήστης. Μπορούμε να το κάνουμε και χωρίς να ορίσουμε κάποια μεταβλητή. Σε κανονικές συνθήκες αυτό θα ήταν ένα infinite loop, αλλά ευτυχώς υπάρχει η break.

4. Ας ενημερώσουμε το χρήστη για το τι έχει επιλέξει.

5. Και μην ξεχάσουμε να πάρουμε την είσοδο.

- Καλό θα είναι όμως να μπορούμε να κάνουμε και αλλαγές στο καλάθι μας.

```
>>> ================================ RESTART ================================
>>>
Please enter an item:
apples
Please enter an item:
oranges
Please enter an item:
pineapples
Please enter an item:

['apples', 'oranges', 'pineapples']
Finished!
>>>
```

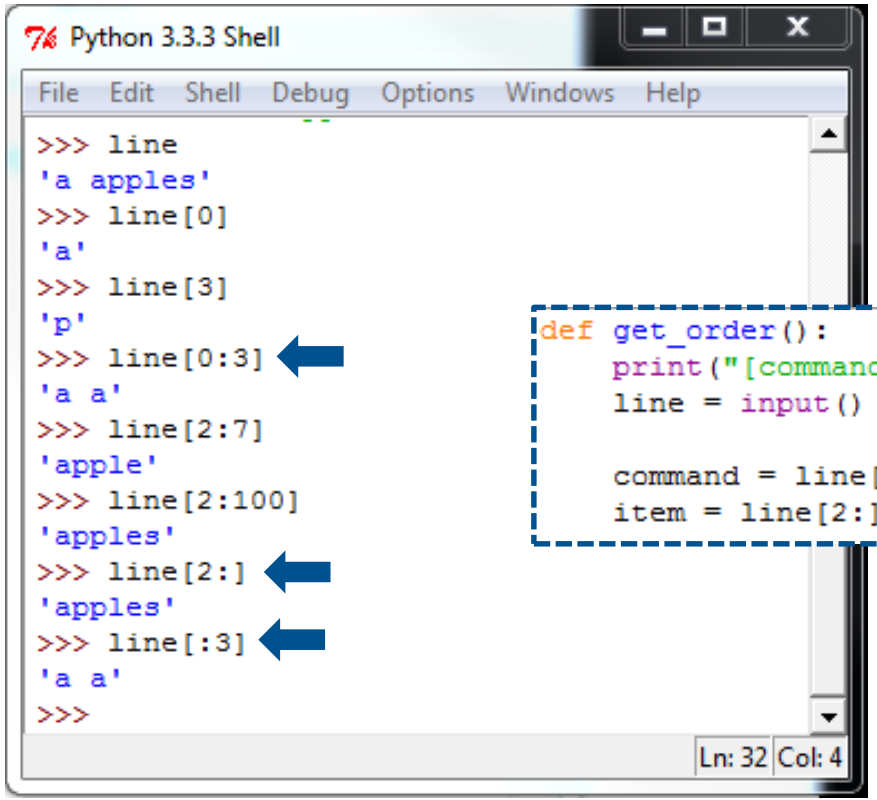
- Ας πάμε όμως σε ένα μεγαλύτερο επίπεδο αφαίρεσης.
- Θα ήθελα όχι απλά να παίρνω αντικείμενα από το χρήστη, αλλά να μπορώ να προσθέτω ή να αφαιρώ αντικείμενα…

```
def get_item():
    print("Please enter an item: ")
    return input()


def go_shopping():
    cart = []

    while True:
        item = get_item()
        if item == "":
            break
        cart.append(item)

    print(cart)
    print("Finished!")


go_shopping()
```

```
def get_order():
    print("Please enter an item: ")
    return input()


def go_shopping():
    cart = []

    while True:
        item = get_order()
        if item == "":
            break
        cart.append(item)

    print(cart)
    print("Finished!")


go_shopping()
```

Τεμαχίζοντας την παραγγελία…

- Για να το κάνω αυτό θα πρέπει να μου πει ο χρήστης κάθε φορά το τι θέλει να κάνει.
- Ας πάρουμε μία μικρή βοήθεια από το Python Shell.
- Παρατηρείστε ότι μπορώ να καθορίσω όποιους από τους χαρακτήρες της λέξης θέλω.




```
def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]
```

Πλειάδες - Tuples

```
>>> line = "a apples"
>>> line
'a apples'
>>> command = line[:1]
>>> command
'a'
>>> item = line[:2]
>>> item
'a '
>>> order = command, item
>>> order
('a', 'a ')
>>> line = "a apples"
>>> line
'a apples'
>>> command = line[:1]
>>> command
'a'
>>> item = line[2:]
>>> item
'apples'
>>> order = command, item
>>> order
('a', 'apples')
>>> order = (command, item)
>>> order
('a', 'apples')
```

- Μέχρι τώρα είδαμε συμβολοσειρές και λίστες που στην Python ονομάζονται τύποι ακολουθίας (sequence types) γιατί με αυτές μπορούμε να διαχειριστούμε συλλογές στοιχείων.
- Ας δούμε έναν ακόμα τύπο ακολουθίας που είναι δημοφιλής.
- Τις πλειάδες.
- Και αφού δε μπορώ να κάνω τα ίδια πράγματα με τις λίστες, γιατί να επιλέγω τις πλειάδες;
  - Οι πλειάδες είναι γενικά πιο γρήγορες και πιο ασφαλείς.

```
>>> len(order)
2
>>> "apples" in order
True
>>> order[0]
'a'
>>> order[1]
'apples'
>>> order.append("something")
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    order.append("something")
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

Εδώ διαφέρει σε σχέση με τις λίστες

Και μπορούμε να κάνουμε και πολλά ακόμα με τις πλειάδες...

```
def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]

    retutn command, item

def go_shopping():
    cart = []

    while True:
        item = get_order()
        if item == "":
            break
        cart.append(item)

    print(cart)
    print("Finished!")


go_shopping()
```

- Προσέξτε ότι με τις δύο πλειάδες μπορούμε να επιστρέψουμε αυτό που αρχικά θέλαμε…

```
>>> order = (command, item)
>>> order
('a', 'apples')
>>> len(order)
2
>>> "apples" in order
True
>>> order[0]
'a'
>>> order[1]
'apples'
>>> order.append("something")
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    order.append("something")
AttributeError: 'tuple' object has no attribute 'append'
>>> c, i = order
>>> c
'a'
>>> i
'apples'
>>>
```

Tuple unpacking

Προχωράμε με τη διαχείριση της παραγγελίας

```
def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]

    return command, item

def process_order(order, cart):
    command, item = order

    if command == "a":
        cart.append(item)
    elif command == "d":
        cart.remove(item)
    elif command == "q":
        return False

    return True

def go_shopping():
    cart = []

    while True:
        order = get_order()
        if not process_order(order, cart):
            break


    print(cart)
    print("Finished!")


go_shopping()
```

- Με τον τρόπο αυτό διαχειριζόμαστε την είσοδο από το χρήστη.
- Αλλάξαμε τον κώδικα στο συγκεκριμένο σημείο για να διαχειριστούμε την περίπτωση που η `process_order()` επιστρέφει `False`.

```
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
a cherries
[command] [item] (command is a to add, d to delete, q to quit)
d cherries
[command] [item] (command is a to add, d to delete, q to quit)
q
['apples', 'oranges']
Finished!
>>>
```

Τι θα συμβεί όμως αν πάμε να σβήσουμε κάτι που δεν υπάρχει στη λίστα;

```
>>> go_shopping()
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
d cherries
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    go_shopping()
  File "C:\examples\python_shopping_cart\shopping.py", line 27, in go_shopping
    if not process_order(order, cart):
  File "C:\examples\python_shopping_cart\shopping.py", line 16, in process_order
    cart.remove(item)
ValueError: list.remove(x): x not in list
>>>
```

- Προφανώς θα προκύψει λάθος και καλό θα είναι να το διορθώσουμε

```
def process_order(order, cart):
    command, item = order

    if command == "a":
        cart.append(item)
    elif command == "d" and item in cart:
        cart.remove(item)
    elif command == "q":
        return False

    return True
```

```
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a oranges
[command] [item] (command is a to add, d to delete, q to quit)
d apples
[command] [item] (command is a to add, d to delete, q to quit)
d apples
[command] [item] (command is a to add, d to delete, q to quit)
q
['oranges']
Finished!
>>>
```

Τι θα συμβεί αν προσθέσουμε δύο ίδια αντικείμενα στη λίστα

```
>>> go_shopping()
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
q
['apples', 'apples']
Finished!
>>>
```

```
>>> alist = [1, 2, 3]
>>> atuple = (1, 2, 3)
>>> aset = {1, 2, 3}
>>> aset
{1, 2, 3}
>>> alist.append(0)
>>> alist
[1, 2, 3, 0]
>>> aset.add(0)
>>> aset
{0, 1, 2, 3}
>>>
```

- Καλό θα είναι να μη συμβαίνει αυτό.
- Μέχρι τώρα είδαμε δύο δομές δεδομένων τις λίστες και τις πλειάδες. Μία βασική διαφορά τους ήταν ότι δε μπορούμε να αλλάξουμε τις πλειάδες, δεν μπορούμε να προσθέσουμε ή να αφαιρέσουμε στοιχεία.
- Ας δούμε μία ακόμα δομή. Το σύνολο (set). Μοιάζει περισσότερο με τη λίστα. Και στα σύνολα μπορούμε να προσθέσουμε στοιχεία. Αλλά έχει και διαφορές!
- Στο σύνολο δε μπορούμε να έχουμε διπλές τιμές. Και αυτό είναι πολύ χρήσιμο!

Ας αξιοποιήσουμε τη νέα δυνατότητα για να διορθώσουμε το πρόγραμμά μας

```
def get_order():
    print("[command] [item] (command is a to add, d to delete, q to quit)")
    line = input()

    command = line[:1]
    item = line[2:]

    return command, item

def process_order(order, cart):
    command, item = order

    if command == "a":
        cart.add(item)
    elif command == "d" and item in cart:
        cart.remove(item)
    elif command == "q":
        return False

    return True

def go_shopping():
    cart = set()

    while True:
        order = get_order()
        if not process_order(order, cart):
            break


    print(cart)
    print("Finished!")


go_shopping()
```

- Αλλάζοντας απλά το `cart` από list σε set πετυχαίνουμε αυτό που θέλουμε.

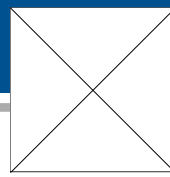
```
>>> ================================ RESTART ================================
>>>
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
a apples
[command] [item] (command is a to add, d to delete, q to quit)
q
{'apples'}
Finished!
>>>
```

Διαφορές Set από List

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> len(s1)
3
>>> s1[0]
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    s1[0]
TypeError: 'set' object does not support indexing
>>> 1 in s1
True
>>> s1.difference(s2)
{1}
>>> s1 - s2
{1}
>>> s2 - s1
{4}
>>> s1 & s2
{2, 3}
>>> s1 ^ s2
{1, 4}
```

- Το set είναι unordered.
- Τα στοιχεία του είναι μοναδικά.
- Μας επιτρέπει να κάνουμε πράξεις συνόλων (π.χ. unions, differences)

Και στην περίπτωση που θέλουμε να αγοράσουμε δύο σακούλες μήλα;

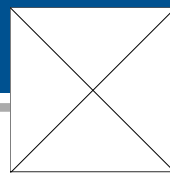


```
>>> adict = { "tzimas" : "tzimas@cti.gr", "tsaknakis" : "tsaknaki@ceid.upatras.gr"}
>>> adict
{'tsaknakis': 'tsaknaki@ceid.upatras.gr', 'tzimas': 'tzimas@cti.gr'}
>>> len(adict)
2
>>> adict[0]
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    adict[0]
KeyError: 0
>>> adict["tzimas"]
'tzimas@cti.gr'
>>> for key in adict:
        print(key)


tsaknakis
tzimas
>>> for key in adict:
        print(adict[key])


tsaknaki@ceid.upatras.gr
tzimas@cti.gr
>>> for key in adict:
        print(key, adict[key])


tsaknakis tsaknaki@ceid.upatras.gr
tzimas tzimas@cti.gr
>>> adict["count"] = 4
>>> adict
{'tsaknakis': 'tsaknaki@ceid.upatras.gr', 'tzimas': 'tzimas@cti.gr', 'count': 4}
>>> adict[3] = 5
>>> adict
{3: 5, 'tsaknakis': 'tsaknaki@ceid.upatras.gr', 'tzimas': 'tzimas@cti.gr', 'count':
4}
```

- Ας δούμε μία ακόμα ενδιαφέρουσα δομή της Python, το λεξικό (dictionary).
- Κάθε στοιχείο ενός λεξικού έχει ένα κλειδί και μία συσχετιζόμενη τιμή.
  `key : value`
- Δεν υπάρχει δεικτοδότηση σε ένα λεξικό, αλλά μπορούμε να το διατρέξουμε.

Και φυσικά μπορούμε να προσθέσουμε αντικείμενα.

Ας αξιοποιήσουμε τη νέα δομή στο πρόγραμμά μας...



```
>>> cart = {"apples" : 1}
>>> cart
{'apples': 1}
>>> cart["apples"] += 1
>>> cart
{'apples': 2}
>>> cart["apples"] -= 1
>>> cart
{'apples': 1}
>>> del cart["apples"]
>>> cart
{}
>>>
```

- Προσέξτε ότι μπορούμε να αυξομοιώσουμε τους δείκτες μας και να διαγράψουμε ένα στοιχείο από το λεξικό.

Τι είδαμε μέχρι τώρα;

- Φτιάξαμε ένα καλάθι αγορών και μέσα από αυτή τη διαδικασία είδαμε δομές δεδομένων που υποστηρίζονται από την Python.
  - Lists,
  - Tuples,
  - Sets και
  - Dictionaries.

Όνομα	Κατασκευή	Περιγραφή
List	[] ή list()	Μία συλλογή αντικειμένων με δυνατότητα ταξινόμησης και αλλαγής μεγέθους.
Tuple	() ή tuple()	Μία αμετάβλητη συλλογή.
Set	{} ή set()	Μία συλλογή μοναδικών αντικειμένων.
Dictionary	{} ή dict()	Μία συλλογή με ζευγάρια ονόματος τιμής.