1. Γράψτε ένα πρόγραμμα το οποίο κατασκευάζει μια λίστα με τους διαιρέτες ενός φυσικού αριθμού.

   Απάντηση: Το πρόγραμμά μας ζητάει από το χρήστη ένα θετικό ακέραιο και αποθηκεύει τους διαιρέτες του, εκτός της μονάδας και του ίδιου του αριθμού, στη λίστα `divisors`:

   ```
   n = int(input('Give a positive integer: '))

   divisors = []
   for i in range(2, n):
       if n%i == 0: divisors.append(i)

   print('The divisors of', n, 'are:', divisors)
   ```

2. Γράψτε ένα πρόγραμμα το οποίο ελέγχει αν δυο λίστες έχουν τουλάχιστον ένα κοινό στοιχείο.

   Απάντηση: Αρκεί να ελέγξουμε αν κάποιο από τα στοιχεία της μιας λίστας ανήκει στην άλλη λίστα ή όχι. Χρησιμοποιούμε τη λογική μεταβλητή `found` η οποία παίρνει την τιμή `True` μόνο αν βρεθεί κοινό στοιχείο.

   ```
   L1 = [11, 12, 13, 10, 9, 8]
   L2 = [7, 8, 10, 13, 9, 14]

   found = False
   for x in L1:
       if x in L2:
           found = True
           break

   if found:
       print('Yes')
   else:
       print('No')
   ```

3. Γράψτε ένα πρόγραμμα το οποίο αποθηκεύει σε μια λίστα τις λέξεις που εισάγει ο χρήστης και στη συνέχεια εμφανίζει μόνο τις λέξεις με περισσότερα από 3 γράμματα. Η είσοδος των λέξων θα πρέπει να τερματίζει όταν ο χρήστης δώσει τη λέξη 'end'.

   Απάντηση: Το πρόβλημα χωρίζεται λογικά σε δύο μέρη. Στο πρώτο διαβάζουμε τις λέξεις που δίνει ο χρήστης μέχρι να συνατήσουμε τη λέξη `end` και στο δεύτερο μέρος τυπώνουμε τις λέξεις με περισσότερα από τρία γράμματα.

   ```
   words = []
   while True:
       word = input('Enter a word: ')
       if word == 'end':
           break
       else:
           words.append(word)

   for word in words:
       if len(word) > 3: print(word)
   ```

4. Γράψτε ένα πρόγραμμα το οποίο αποθηκεύει στη λίστα L τις τιμές της συνάρτησης f στα σημεία ενός ομοιόμορφου διαμερισμού του διαστήματος $[0, 1]$ με n εσωτερικά σημεία.

   Απάντηση: Παρατηρούμε κατ' αρχήν ότι n σημεία (ανά δύο διαφορετικά μεταξύ τους) στο εσωτερικό κάποιου διαστήματος $[a, b]$ το χωρίζουν σε $n + 1$ διαστήματα. Αν θέλουμε όλα αυτά τα διαστήματα να έχουν το ίδιο μήκος, τότε αυτό πρέπει να είναι $(b - a)/(n + 1)$.

```
def f(x):
    return x**2 + 6*x - 5

a = 0
b = 1
n = 15
d = (b - a) / (n + 1)

values = []
for i in range(n+2):
    values.append( f(a + i*d) )
```

5. Γράψτε μια συνάρτηση η οποία με είσοδο δύο λίστες τα στοιχεία των οποίων είναι σε αύξουσα σειρά επιστρέφει μια λίστα με όλα τα στοιχεία και από τις δύο λίστες, πάλι σε αύξουσα σειρά.

   Απάντηση: Μια λύση στο συγκεκριμένο πρόβλημα θα ήταν να ενώσουμε τις δύο λίστες και μετά να εφαρμόσουμε τη μέθοδο `sort`. Μια (καλύτερη;) ιδέα είναι αυτή που υλοποιείται παρακάτω:

```
L1 = [2, 4, 7, 9, 12, 17]
L2 = [3, 4, 5, 6, 19]

L = []
i = 0
j = 0

while True:
    if i == len(L1):
        L.extend(L2[j:])
        break
    if j == len(L2):
        L.extend(L1[i:])
        break

    if L1[i] <= L2[j]:
        L.append(L1[i])
        i += 1
    else:
        L.append(L2[j])
        j += 1
```

6. Γράψτε μια συνάρτηση η οποία υπολογίζει τη συχνότητα εμφάνισης όλων των στοιχείων μιας λίστας.

   Απάντηση: Το μόνο θέμα με τη συγκεκριμένη άσκηση είναι να μετρήσουμε ακριβώς μια φορά τη συχνότητα εμφάνισης των στοιχείων που εμφανίζονται περισσότερο από μία φορές. Για το λόγο αυτό διατηρούμε μια λίστα `seen` η οποία θυμάται τα στοιχεία που εμφανίζονται περισσότερο από μια φορές.

```
L = [10, 12, 13, 13, 10, 9, 8, 7, 8, 10, 10, 8, 13]
seen = []
freq = []

for x in L:
    if x not in seen:
        freq.append( L.count(x) )
        seen.append(x)
```

```
for i in range(len(seen)):
    print('Elenent', seen[i], 'appears', freq[i], 'times')
```

7. Γράψτε ένα πρόγραμμα το οποίο αφαιρεί από μια λίστα τις πολλαπλές εμφανίσεις των στοιχείων της. Για παράδειγμα, η λίστα `[10, 12, 13, 13, 10, 9, 8, 7, 8, 10, 10, 8, 13]` θα πρέπει να γίνει `[10, 12, 13, 9, 8, 7]`

   Απάντηση: Το πρόβλημα αυτό είναι παρόμοιο με το προηγούμενο. Διατηρούμε μια λίστα `seen` η οποία θυμάται τα στοιχεία που εμφανίζονται παραπάνω από μία φορές.

```
L = [10, 12, 13, 13, 10, 9, 8, 7, 8, 10, 10, 8, 13]

seen = []
for x in L:
    if x not in seen:
        seen.append(x)

print(seen)
```

8. Μπορούμε να βρούμε το μικρότερο στοιχείο μιας λίστας χρησιμοποιώντας τη συνάρτηση `min`. Πως μπορούμε να βρούμε το *δεύτερο* μικρότερο στοιχείο μιας λίστας; Η άσκηση είναι δύσκολη – αν κάποιος ενδιαφέρεται για μια κομψή λύση.

   Απάντηση: Θα διατηρήσουμε δύο μεταβλητές, τις `frst` και `scnd` οι οποίες θα αποθηκεύουν το μικρότερο και το δεύτερο μικρότερο στοιχείο της λίστας. Κάθε φορά που συναντάμε ένα στοιχείο θα το συγκείνουμε με την τρέχουσα τιμή τόσο της `frst` όσο και `scnd`. Η λύση που προτείνεται παρακάτω είναι μεν κομψή αλλά κάνει περισσότερες συγκρίσεις απ' ότι είναι απαραίτητο.

```
L = [54, 26, 93, 17, 77, 31, 44, 55, 20]

frst = L[0]
scnd = L[0]

for x in L:
    if x < frst:
        scnd = frst
        frst = x
    elif x < scnd:
        scnd = x

print('The second smallest element is', scnd)
```

9. Ο απλούστερος (και ο πιο αργός) τρόπος ταξινόμησης είναι ο λεγόμενος αλγόριθμος της φυσαλίδας (bubble sort), μια υλοποίηση του οποίου θα μπορούσε να είναι η ακόλουθη:

```
def bubbleSort(L):
    for n in range(len(L)-1,0,-1):
        for i in range(n):
            if L[i] > L[i+1]:
                tmp = L[i]; L[i] = L[i+1]; L[i+1] = tmp

L = [54, 26, 93, 17, 77, 31, 44, 55, 20]
bubbleSort(L)
print(L)
```

Προσπαθήστε να καταλάβετε την ιδέα του αλγορίθμου τυπώνοντας τα περιεχόμενα της λίστας L μετά από κάθε ανακύκλωση. Μετρήστε επίσης τον αριθμό των συγκρίσεων που κάνει η μέθοδος. Μπορείτε να ανακαλύψετε κάποια σχέση με το μήκος της λίστας;

Απάντηση: Είναι φανερό ότι η πρώτη εσωτερική ανακύκλωση τοποθετεί στην τελευταία θέση της λίστας το μεγαλύτερο στοιχείο της, κάνοντας στη πορεία $n-1$ συγκρίσεις στοιχείων. Η δεύτερη εσωτερική ανακύκλωση τοπθετεί στην προτελευταία θέση της λίστας το δεύτερο μεγαλύτρεο στοιχείο της, κάνονττας $n-2$ συγκρίσεις. Συνεχίζοντας με τον ίδιο τρόπο βλέπουμε ότι ο αλγόριθμος της φυσαλίδας κάνει συνολικά $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$ συγκρίσεις.