ΠΛΗΡΟΦΟΡΙΚΗ Ι (Python)
Ενότητα 5

Λεξικά (Dictionaries)

Τα λεξικά είναι μια μεταβλητή δομή δεδομένων της Python για την αποθήκευση συλλογών δεδομένων της μορφής: *κλειδί – τιμή*. Δηλαδή, κάθε στοιχείο σε ένα λεξικό έχει δύο μέρη: ένα κλειδί και μία τιμή. Το κλειδί χρησιμοποιείται για τον εντοπισμό μιας συγκεκριμένης τιμής. Στην πράξη, τα στοιχεία ενός λεξικού συνήθως αναφέρονται ως ζεύγη κλειδιού-τιμής. Για την ανάκτηση μιας συγκεκριμένης τιμής από ένα λεξικό, χρησιμοποιείται το αντίστοιχο κλειδί που σχετίζεται με την τιμή αυτή.

Ουσιαστικά, ένα λεξικό είναι ένα μη-διατεταγμένο σύνολο από ζεύγη κλειδιού-τιμής. Οι τιμές δεν έχουν δείκτες όπως σε μια λίστα, αλλά μοναδικά κλειδιά, τα οποία χρησιμεύουν για τον εντοπισμό τους (ή την αναφορά, πρόσβαση κτλ. σε αυτές). Τα κλειδιά μπορούν να είναι οποιοσδήποτε αμετάβλητος (immutable) τύπος, π.χ. αριθμοί, συμβολοσειρές ή πλειάδες (αρκεί οι πλειάδες να μην περιέχουν κάποιον μεταβλητό τύπο).

Δημιουργία Λεξικού

Τα λεξικά δημιουργούνται περικλείοντας μέσα σε άγκιστρα, ζευγάρια της μορφής `κλειδί:τιμή` χωρισμένα μεταξύ με κόμμα:

```
λεξικό = {κλειδί_1:τιμή_1, κλειδί_2:τιμή_2, ...}
```

Οι εντολές:

```
λεξικό = {}
```

και

```
λεξικό = dict()
```

δημιουργούν κενό λεξικό. Με την ενσωματωμένη συνάρτηση `dict` υπάρχει και η δυνατότητα δημιουργίας λεξικού με συγκεκριμένα στοιχεία, ως εξής:

```
λεξικό = dict([(κλειδί_1, τιμή_1), (κλειδί_2, τιμή_2), ...])
```

δηλαδή τα δύο μέρη του κάθε στοιχείου του λεξικού (κλειδί-τιμή) δίνονται μέσα σε πλειάδα και όλες οι πλειάδες αποτελούν στοιχεία λίστας, την οποία η συνάρτηση `dict` μετατρέπει σε λεξικό. Π.χ.,

```
>>> grades = {'Student 1':8, 'Student 2':3, 'Student 3':6, 'Student 4':10}
>>> grades
{'Student 3': 6, 'Student 2': 3, 'Student 1': 8, 'Student 4': 10}
>>> prices = dict([('Item 1', 19), ('Item 2', 25.5), ('Item 3', 40)])
>>> prices
{'Item 3': 40, 'Item 1': 19, 'Item 2': 25.5}
```

Προσπέλαση τιμής λεξικού

Τα στοιχεία των λεξικών *δεν* είναι αποθηκευμένα με κάποια σειρά, όπως είναι στις λίστες. Άρα δεν υπάρχει η έννοια του δείκτη που υπάρχει στις λίστες. Αντί για αυτό, η τιμή του κάθε στοιχείου προσδιορίζεται από το κλειδί της. Επομένως, η πρόσβαση σε κάποιο στοιχείο γίνεται ως εξής:

```
λεξικό[κλειδί]
```

Αν το συγκεκριμένο κλειδί δεν υπάρχει στο λεξικό, τότε προκαλείται μια εξαίρεση. Π.χ.,

```
>>> grades = {'Student 1':8, 'Student 2':3, 'Student 3':6, 'Student 4':10}
>>> grades
{'Student 3': 6, 'Student 2': 3, 'Student 1': 8, 'Student 4': 10}
>>> grades['Student 2']
3
>>> grades['Student 5']
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    grades['Student 5']
KeyError: 'Student 5'
```

Η επαναληπτική προσπέλαση των στοιχείων ενός λεξικού μπορεί να γίνει με έναν βρόχο `for` ως εξής:

```
for μεταβλητή in λεξικό:
    εντολή
    εντολή
    ...
```

όπου η *μεταβλητή* αναφέρεται σε κάθε *κλειδί* του λεξικού. Π.χ.,

```
>>> for i in grades:
      print(i, grades[i])

Student 3 6
Student 2 3
Student 1 8
Student 4 10
```

Τελεστές `in` και `not in`

Για να εξασφαλιστεί ότι δεν θα προκληθεί κάποια εξαίρεση σε ένα πρόγραμμα που αναζητάει στοιχεία ενός λεξικού, μπορεί να χρησιμοποιηθεί ο τελεστής `in` για να ελεγχθεί εάν όντως υπάρχει το προς αναζήτηση κλειδί στο λεξικό, ως εξής:

```
if κλειδί in λεξικό:
```

Π.χ.,

```
>>> if 'Student 3' in grades:
      print(grades['Student 3'])
6
```

Υπάρχει επίσης ο τελεστής `not in` που χρησιμοποιείται για να διαπιστωθεί εάν κάποιο κλειδί δεν υπάρχει στο λεξικό, ως εξής:

```
if κλειδί not in λεξικό:
```

Π.χ.,

```
>>> if 'Student 5' not in grades:
      print(Το κλειδί Student 5 δε βρέθηκε.)
Το κλειδί Student 5 δε βρέθηκε.
```

Προσθήκη και τροποποίηση στοιχείων σε λεξικό

Τα λεξικά είναι μεταβλητές δομές δεδομένων. Για την προσθήκη ενός νέου στοιχείου ή την τροποποίηση της τιμής ενός υπάρχοντος στοιχείου, χρησιμοποιείται μια εντολή εκχώρησης της μορφής:

```
λεξικό[κλειδί] = τιμή
```

Αν το *κλειδί* δεν υπάρχει ήδη στο *λεξικό*, δημιουργείται νέο στοιχείο. Αν το *κλειδί* υπάρχει ήδη στο *λεξικό*, τότε η τιμή του αλλάζει σε *τιμή*.

Π.χ.,

```
>>> grades = {'Student 1':8, 'Student 2':3, 'Student 3':6, 'Student 4':10}
>>> grades
{'Student 4': 10, 'Student 3': 6, 'Student 2': 3, 'Student 1': 8}
>>> grades['Student 2'] = 5
>>> grades
{'Student 4': 10, 'Student 3': 6, 'Student 2': 5, 'Student 1': 8}
>>> grades['Student 5'] = 9
>>> grades
{'Student 4': 10, 'Student 3': 6, 'Student 2': 5, 'Student 1': 8, 'Student 5': 9}
```

Διαγραφή στοιχείων από λεξικό

Για τη διαγραφή στοιχείων από ένα λεξικό, χρησιμοποιείται η εντολή `del`, ως εξής:

```
del λεξικό[κλειδί]
```

η οποία διαγράφει το στοιχείο με το συγκεκριμένο *κλειδί*. Αν το *κλειδί* δεν υπάρχει στο *λεξικό*, προκαλείται μια εξαίρεση. Π.χ.,

```
>>> grades
{'Student 4': 10, 'Student 3': 6, 'Student 2': 5, 'Student 1': 8, 'Student 5': 9}
>>> del grades['Student 4']
>>> grades
{'Student 3': 6, 'Student 2': 5, 'Student 1': 8, 'Student 5': 9}
>>> del grades['Student 6']
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    del grades['Student 6']
KeyError: 'Student 6'
```

Και σε αυτή την περίπτωση, θα μπορούσε να χρησιμοποιηθεί ο τελεστής in με ένα if για να πραγματοποιηθεί η διαγραφή μόνο εάν το συγκεκριμένο κλειδί υπάρχει στο λεξικό, έτσι ώστε αν δεν υπάρχει να μη δημιουργηθεί σφάλμα:

```
if 'Student 6' in grades:
    del grades['Student 6']
```

Το πλήθος των στοιχείων ενός λεξικού δίνεται από τη συνάρτηση `len`, όπως και στις λίστες:

`len(`*λεξικό*`)`

Π.χ.,

```
>>> len(grades)
4
```

Οι τιμές των στοιχείων ενός λεξικού μπορούν να είναι αντικείμενα οποιουδήποτε τύπου (ενώ τα κλειδιά πρέπει να είναι αμετάβλητοι τύποι). Επίσης, ένα λεξικό μπορεί να περιέχει τιμές διαφορετικών τύπων. Στο παρακάτω παράδειγμα, δημιουργείται ένα λεξικό με δύο στοιχεία με τιμές λίστες και στη συνέχεια προστίθεται ένα τρίτο στοιχείο με τιμή πλειάδα:

```
>>> grades = {'Student1':[5, 6, 9, 10], 'Student2':[4, 3, 7, 6]}
>>> grades
{'Student1': [5, 6, 9, 10], 'Student2': [4, 3, 7, 6]}
>>> grades['Student3'] = (3, 6, 9, 5)
>>> grades
{'Student1': [5, 6, 9, 10], 'Student3': (3, 6, 9, 5), 'Student2': [4, 3, 7, 6]}
```

Μέθοδοι για λεξικά

Η Python περιέχει κάποιες μεθόδους που χρησιμοποιούνται για τη διαχείριση των λεξικών. Οι πιο σημαντικές είναι οι εξής:

- `items()` → Επιστρέφει όλα τα κλειδιά ενός λεξικού μαζί με τις αντίστοιχες τιμές τους ως μια αλληλουχία από πλειάδες. Ο τύπος αυτός αλληλουχίας ονομάζεται “προβολή λεξικού”.
- `values()` → Επιστρέφει όλες τις τιμές του λεξικού ως μια αλληλουχία τύπου “προβολής λεξικού” (δηλαδή ως μια αλληλουχία από πλειάδες).
- `keys()` → Επιστρέφει όλα τα κλειδιά ενός λεξικού ως μια αλληλουχία τύπου “προβολής λεξικού”.
- `get(`*κλειδί*`, `*προεπιλογή*`)` → Επιστρέφει την τιμή που αντιστοιχεί σε ένα συγκεκριμένο κλειδί. Αν το *κλειδί* δεν υπάρχει, η μέθοδος δεν προκαλεί κάποια εξαίρεση, αλλά επιστρέφει μια προκαθορισμένη τιμή (*προεπιλογή*).
- `pop(`*κλειδί*`, `*προεπιλογή*`)` → Επιστρέφει την τιμή που αντιστοιχεί σε ένα συγκεκριμένο κλειδί και αφαιρεί αυτό το ζεύγος κλειδιού-τιμής από το λεξικό. Αν το *κλειδί* δεν υπάρχει, επιστρέφει μια προκαθορισμένη τιμή (*προεπιλογή*).
- `popitem()` → Επιστρέφει ως πλειάδα ένα τυχαία επιλεγμένο ζεύγος κλειδιού-τιμής και το αφαιρεί από το λεξικό. Μπορεί να εκχωρηθεί σε δύο μεταβλητές, όπου στην πρώτη θα εκχωρηθεί το κλειδί και στη δεύτερη η τιμή.
- `clear()` → Διαγράφει το περιεχόμενο ενός λεξικού.

Παραδείγματα:

```
>>> grades = {'Student 1':8, 'Student 2':3, 'Student 3':6}
>>> all_items = grades.items()
>>> all_items
dict_items([('Student 3', 6), ('Student 1', 8), ('Student 2', 3)])
>>> for key, value in grades.items():
     print(key, value)

Student 3 6
Student 1 8
Student 2 3
>>> all_values = grades.values()
>>> all_values
dict_values([6, 8, 3])
>>> for val in grades.values():
     print(val)

6
8
3
>>> all_keys = grades.keys()
>>> all_keys
dict_keys(['Student 3', 'Student 1', 'Student 2'])
>>> for k in grades.keys():
     print(k)

Student 3
Student 1
Student 2
>>> grade1 = grades.get('Student 1', 'Unknown student')
>>> grade1
8
>>> grade4 = grades.get('Student 4', 'Unknown student')
>>> grade4
'Unknown student'
>>> grade1 = grades.pop('Student 1', 'Unknown student')
>>> grade1
8
>>> grades
{'Student 3': 6, 'Student 2': 3}
>>> grades['Student 4'] = 10
>>> grades
{'Student 3': 6, 'Student 4': 10, 'Student 2': 3}
>>> student, grade = grades.popitem()
>>> print(student, grade)
Student 3 6
>>> grades
{'Student 4': 10, 'Student 2': 3}
>>> grades.clear()
>>> grades
{}
```

Σύνολα (Sets)

Τα σύνολα είναι μια μεταβλητή δομή δεδομένων της Python για την αποθήκευση συλλογών *μοναδικών* (μη επαναλαμβανόμενων) δεδομένων, όπως δηλαδή στα σύνολα στα μαθηματικά. Τα βασικά χαρακτηριστικά των συνόλων είναι τα εξής:

- Όλα τα στοιχεία ενός συνόλου είναι μοναδικά.
- Τα στοιχεία ενός συνόλου δεν είναι διατεταγμένα.
- Τα στοιχεία ενός συνόλου είναι αμετάβλητοι τύποι, όπως τα κλειδιά ενός λεξικού.
- Ένα σύνολο μπορεί αν περιέχει στοιχεία διαφορετικών (αμετάβλητων) τύπων.

Δημιουργία Συνόλου

Τα σύνολα δημιουργούνται με την ενσωματωμένη συνάρτηση `set`:

```
σύνολο = set()
```

Η εντολή αυτή δημιουργεί ένα κενό σύνολο. Μπορούν επίσης να δημιουργηθούν με τη χρήση άγκιστρων, με συγκεκριμένες τιμές:

```
σύνολο = {τιμή_1, τιμή_2, τιμή_3, ...}
```

Κενό σύνολο μπορεί να δημιουργηθεί μόνο με τη συνάρτηση `set` όπως προαναφέρθηκε, γιατί η χρήση κενών άγκιστρων (`{}`) δημιουργεί λεξικό και όχι σύνολο.

Εάν κατά τη δημιουργία συνόλου με τη χρήση άγκιστρων περιληφθούν πολλαπλά στοιχεία με την ίδια τιμή, αυτά αφαιρούνται και παραμένει ένα μόνο στοιχείο για κάθε τιμή.

Π.χ.,

```
>>> set1 = set()
>>> set1
set()
>>> set2 = {1, 2, 3, 2, 4, 3, 1, 2}
>>> set2
{1, 2, 3, 4}
```

Η συνάρτηση `set` μπορεί να δεχτεί ένα όρισμα για τη δημιουργία συνόλου με συγκεκριμένες τιμές. Το όρισμα αυτό θα πρέπει να είναι κάποια αλληλουχία τιμών, όπως λίστα, πλειάδα ή συμβολοσειρά. Π.χ.,:

```
σύνολο = set([τιμή_1, τιμή_2, τιμή_3, ...])
```

ή

```
σύνολο = set((τιμή_1, τιμή_2, τιμή_3, ...))
```

ή

```
σύνολο = set('συμβολοσειρά')
```

Στην τελευταία περίπτωση, κάθε χαρακτήρας της συμβολοσειράς αποτελεί και ένα στοιχείο του συνόλου. Αυτό δημιουργεί πρόβλημα στην περίπτωση που κάποιος θέλει να δημιουργήσει σύνολο όπου το κάθε στοιχείο του να είναι συμβολοσειρά. Οι εντολές:

```
σύνολο = set('συμβολοσειρά_1', 'συμβολοσειρά_2', ...) # ΣΥΝΤΑΚΤΙΚΟ ΣΦΑΛΜΑ
```

και

```
σύνολο = set('συμβολοσειρά_1 συμβολοσειρά_2 ...')      # ΛΟΓΙΚΟ ΣΦΑΛΜΑ
```

δεν έχουν το επιθυμητό αποτέλεσμα (δημιουργία συνόλου με στοιχεία `'συμβολοσειρά_1'`, `'συμβολοσειρά_2'` κτλ), γιατί: η 1η εντολή είναι συντακτικά λάθος αφού η συνάρτηση `set` έχει μία μόνο παράμετρο εισόδου, άρα πρέπει να κληθεί με ένα μόνο όρισμα, ενώ η 2η εντολή δημιουργεί σύνολο με στοιχεία τους χαρακτήρες ολόκληρης της συμβολοσειράς που περικλείεται στα εισαγωγικά. Η σωστή εντολή είναι η εξής:

```
σύνολο = set(['συμβολοσειρά_1', 'συμβολοσειρά_2', ...])
```

δηλαδή το όρισμα της `set` είναι μια λίστα με στοιχεία τις συμβολοσειρές που θα αποτελέσουν στοιχεία του συνόλου.

Π.χ.,

```
>>> set3 = set([1, 2, 3, 4, 5])
>>> set4 = set((6, 7, 8, 9, 10))
>>> set3
{1, 2, 3, 4, 5}
>>> set4
{8, 9, 10, 6, 7}
>>> set5 = set([1, 2, 3, 2, 3, 4])
>>> set5
{1, 2, 3, 4}
>>> set6 = set(['Hello', 0, 1])
>>> set6
{0, 1, 'Hello'}
>>> set7 = set('Hello')
>>> set7
{'H', 'o', 'l', 'e'}
>>> set8 = set('Hello there')
>>> set8
{'r', 'l', 'e', ' ', 'o', 'h', 't', 'H'}
>>> set9 = set(['Hello', 'there'])
>>> set9
{'there', 'Hello'}
```

Το πλήθος των στοιχείων ενός συνόλου δίνεται από τη συνάρτηση `len`, όπως και στις λίστες ή τα λεξικά:

```
len(σύνολο)
```

Π.χ.,

```
>>> len(set9)
2
```

Προσθήκη και διαγραφή στοιχείων συνόλου

Τα σύνολα είναι μεταβλητές δομές δεδομένων. Για την προσθήκη ενός νέου στοιχείου χρησιμοποιείται η μέθοδος `add`:

```
σύνολο.add(στοιχείο)
```

Για την προσθήκη πολλών στοιχείων σε ένα σύνολο, χρησιμοποιείται η μέθοδος `update`, η οποία δέχεται μια αλληλουχία τιμών, π.χ. μια λίστα ή μια πλειάδα ή μια συμβολοσειρά, και προσθέτει ένα-ένα τα στοιχεία της στο σύνολο:

```
σύνολο.update(αλληλουχία_τιμών)
```

Π.χ.,

```
>>> set10 = set([1, 2, 3, 4])
>>> set10
{1, 2, 3, 4}
>>> set10.add(5)
>>> set10
{1, 2, 3, 4, 5}
>>> set10.update([6, 7, 'Hello'])
>>> set10
{1, 2, 3, 4, 5, 6, 7, 'Hello'}
>>> set10.update('there')
>>> set10
{1, 2, 3, 4, 5, 6, 7, 'r', 'Hello', 'e', 'h', 't'}
```

Για τη διαγραφή κάποιου στοιχείου ενός συνόλου χρησιμοποιούνται οι μέθοδοι `remove` και `discard`. Και οι δύο μέθοδοι δέχονται το προς διαγραφή στοιχείο και το αφαιρούν από το σύνολο, με τη διαφορά ότι εάν το στοιχείο δεν υπάρχει στο σύνολο, η μέθοδος `remove` προκαλεί μια εξαίρεση ενώ η μέθοδος `discard` δεν προκαλεί κάποια εξαίρεση. Π.χ.,

```
>>> set11 = {1, 2, 3, 4, 5}
>>> set11
{1, 2, 3, 4, 5}
>>> set11.remove(3)
>>> set11
{1, 2, 4, 5}
>>> set11.discard(4)
>>> set11
{1, 2, 5}
>>> set11.discard(6)
>>> set11.remove(6)
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    set11.remove(6)
KeyError: 6
```

- Προφανώς, η *τροποποίηση* ενός στοιχείου δεν έχει κάποια λογική έννοια στα σύνολα.

Προσπέλαση στοιχείων συνόλου

Για την προσπέλαση όλων των στοιχείων ενός συνόλου, χρησιμοποιείται ο βρόχος `for` ως εξής:

```
for μεταβλητή in σύνολο:
    εντολή
    εντολή
    ...
```

όπου η *μεταβλητή* αναφέρεται σε κάθε στοιχείο του συνόλου. Π.χ.,

```
>>> set11
{1, 2, 5}
>>> for i in set11:
      print(i)

1
2
5
```

Τελεστές `in` και `not in`

Με τον τελεστή `in` μπορεί να ελεγχθεί εάν κάποια τιμή ανήκει σε ένα σύνολο, ενώ με τον τελεστή `not in` εάν δεν ανήκει. Π.χ.,

```
>>> set11
{1, 2, 5}
>>> 2 in set11
True
>>> 3 in set11
False
>>> 3 not in set11
True
```

Πράξεις μεταξύ συνόλων

Υπάρχει η δυνατότητα πραγματοποίησης πράξεων μεταξύ συνόλων, είτε με τη χρήση των κατάλληλων μεθόδων είτε με τη χρήση των αντίστοιχων τελεστών, όπως φαίνεται στον παρακάτω πίνακα.

Πράξη	**Μέθοδος**	**Τελεστής**
Ένωση	`a.union(b)`	`a | b`
Τομή	`a.intersection(b)`	`a & b`
Διαφορά	`a.difference(b)`	`a - b`
Συμμετροδιαφορά	`a.symmetric_difference(b)`	`a ^ b`
Υποσύνολο	`a.issubset(b)`	`a <= b`
Υπερσύνολο	`a.issuperset(b)`	`a >= b`

Παραδείγματα:

```
>>> set1 = {1, 2, 3, 4, 5}
>>> set2 = {1, 2, 3, 6, 7}
>>> set1.union(set2)
{1, 2, 3, 4, 5, 6, 7}
>>> set1.intersection(set2)
{1, 2, 3}
>>> set1.difference(set2)
{4, 5}
>>> set2.difference(set1)
{6, 7}
>>> set1.symmetric_difference(set2)
{4, 5, 6, 7}
>>> set3 = {1, 3, 5, 7}
>>> set3.issubset(set1)
False
>>> set3 = {1, 3, 5}
>>> set3.issubset(set1)
True
>>> set3.issubset(set2)
False
>>> set1.issuperset(set3)
True
```

Συμβολοσειρές (Strings)

Οι συμβολοσειρές ουσιαστικά αποτελούν αλληλουχίες χαρακτήρων. Η Python έχει τη δυνατότητα να χειρίζεται τις συμβολοσειρές με αυτόν τον τρόπο (ως αλληλουχίες χαρακτήρων) και διαθέτει συγκεκριμένες μεθόδους για τη διαχείρισή τους.

Η προσπέλαση στα στοιχεία μιας συμβολοσειράς (δηλαδή στους επιμέρους χαρακτήρες της) μπορεί να γίνει είτε με έναν βρόχο `for`, είτε με τη χρήση δεικτών. Με βρόχο `for`:

```
for μεταβλητή in συμβολοσειρά:
    εντολή
    εντολή
    ...
```

όπου σε κάθε επανάληψη εκχωρείται στη *μεταβλητή* από ένας χαρακτήρας της συμβολοσειράς. Π.χ.,

```
>>> text = 'Hello'
>>> for ch in text:
     print(ch)

H
e
l
l
o
```

Οι συμβολοσειρές είναι μια αμετάβλητη (immutable) δομή δεδομένων, επομένως τα στοιχεία τους δεν μπορούν να τροποποιηθούν. Αυτό φαίνεται π.χ. στο παρακάτω παράδειγμα:

```
>>> text = 'Hello'
>>> for ch in text:
     ch = 'x'
>>> print(text)
Hello
```

δηλαδή η εντολή `ch = 'x'`μέσα στο `for` δεν άλλαξε κάποιο από τα στοιχεία της συμβολοσειράς.

Παράδειγμα:

- Πρόγραμμα που μετράει τις εμφανίσεις κάποιου χαρακτήρα σε μια συμβολοσειρά:

```
def main():
    text = input('Πληκτρολόγησε ένα κείμενο: ')
    t = input('Δώσε έναν χαρακτήρα: ')
    count = 0
    for ch in text:
        if ch==t:
            count += 1
    print('Ο χαρακτήρας', t, 'εμφανίζεται', count, 'φορές στο κείμενο.')
```

Με τη χρήση δεικτών, η προσπέλαση στα στοιχεία μιας συμβολοσειράς μπορεί να γίνει όπως και στις λίστες:

```
συμβολοσειρά[δείκτης]
```

όπου και εδώ, η αρίθμηση των δεικτών ξεκινάει από το 0, ενώ υπάρχουν και οι ίδιες δυνατότητες χρήσης του τελεστή τεμαχισμού : όπως και στις λίστες. Επειδή όμως πρόκειται για αμετάβλητο τύπο δεδομένων, οποιαδήποτε εντολή εκχώρησης τιμής σε κάποιο στοιχείο, προκαλεί σφάλμα. Π.χ.,

```
>>> text = 'Hello'
>>> text[1]
'e'
>>> text[1:3]
'el'
>>> text[1:10]
'ello'
>>> text[1] = 'x'
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    text[1] = 'x'
TypeError: 'str' object does not support item assignment
```

Για τον υπολογισμό του πλήθους των χαρακτήρων μιας συμβολοσειράς, χρησιμοποιείται η συνάρτηση `len`. Π.χ.,

```
>>> text = 'Hello there'
>>> index = 0
>>> while index < len(text):
     print(text[index], end = ' ')
     index += 1

H e l l o   t h e r e
```

Υπάρχει επίσης και εδώ, όπως και στις λίστες, ο τελεστής συνένωσης +, καθώς και ο +=:

```
συμβολοσειρά_1 = συμβολοσειρά_2 + συμβολοσειρά_3
```

ή

```
συμβολοσειρά_1 += συμβολοσειρά_2
```

Επειδή όμως οι συμβολοσειρές είναι αμετάβλητος τύπος, εντολές της μορφής:

```
συμβολοσειρά_1 = συμβολοσειρά_1 + συμβολοσειρά_2
```

ή

```
συμβολοσειρά_1 += συμβολοσειρά_2
```

που φαινομενικά τροποποιούν τη `συμβολοσειρά_1`, δεν την τροποποιούν, αλλά δημιουργούν *νέα* συμβολοσειρά στην οποία αναφέρεται πλέον το όνομα `συμβολοσειρά_1`.

Τελεστές `in` και `not in`

Με τον τελεστή `in` μπορεί να ελεγχθεί εάν μια συμβολοσειρά περιέχεται σε κάποια άλλη συμβολοσειρά, ενώ με τον τελεστή `not in` εάν δεν περιέχεται. Π.χ.,

```
>>> text = 'Hello there'
>>> 'hello' in text
False
>>> 'there' in text
True
>>> 'Hello' not in text
False
```

Μέθοδοι για συμβολοσειρές

Η Python περιέχει πολλές μεθόδους που χρησιμοποιούνται για τη διαχείριση συμβολοσειρών. Οι πιο σημαντικές για τον *έλεγχο συμβολοσειρών* είναι οι εξής:

- `isalnum()` → Επιστέφει `True` αν η συμβολοσειρά περιέχει μόνο γράμματα ή αριθμητικά ψηφία.
- `isalpha()` → Επιστέφει `True` αν η συμβολοσειρά περιέχει μόνο γράμματα.
- `isdigit()` → Επιστέφει `True` αν η συμβολοσειρά περιέχει μόνο αριθμητικά ψηφία.
- `isspace()` → Επιστέφει `True` αν η συμβολοσειρά περιέχει μόνο λευκούς χαρακτήρες (δηλαδή κενά, `\n` και `\t`).
- `islower()` → Επιστέφει `True` αν όλα τα γράμματα της συμβοσειράς είναι πεζά.
- `isupper()` → Επιστέφει `True` αν όλα τα γράμματα της συμβοσειράς είναι κεφαλαία.

Οι πιο σημαντικές μέθοδοι για τη *δημιουργία τροποποιημένων εκδόσεων συμβολοσειρών* είναι οι εξής:

- `lower()` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας μετατρέψει όλα τα γράμματα σε πεζά.
- `upper()` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας μετατρέψει όλα τα γράμματα σε κεφαλαία.
- `strip()` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας αφαιρέσει όλους τους αρχικούς και τελικούς λευκούς χαρακτήρες (δηλαδή από την αρχή και από το τέλος).
- `strip(char)` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας αφαιρέσει όλες τις αρχικές και τελικές εμφανίσεις της συμβολοσειράς `char`.
- `lstrip()` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας αφαιρέσει όλους τους αρχικούς λευκούς χαρακτήρες, δηλαδή όσους βρίσκονται στην αρχή της συμβολοσειράς.
- `lstrip(`*`char`*`)` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας αφαιρέσει όλες τις αρχικές εμφανίσεις της συμβολοσειράς *`char`*, δηλαδή αυτές που βρίσκονται στην αρχή της συμβολοσειράς.
- `rstrip()` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας αφαιρέσει όλους τους τελικούς λευκούς χαρακτήρες, δηλαδή όσους βρίσκονται στο τέλος της συμβολοσειράς.
- `rstrip(`*`char`*`)` → Επιστέφει ένα αντίγραφο της συμβολοσειράς έχοντας αφαιρέσει όλες τις τελικές εμφανίσεις της συμβολοσειράς *`char`*, δηλαδή αυτές που βρίσκονται στο τέλος της συμβολοσειράς.

Οι πιο σημαντικές μέθοδοι για την *αναζήτηση και την αντικατάσταση υπο-συμβολοσειρών σε συμβολοσειρές* είναι οι εξής:

- `find(substring)` → Επιστέφει τον δείκτη της συμβολοσειράς από τον οποίο ξεκινάει η υπο-συμβολοσειρά *`substring`* στην πρώτη της εμφάνιση μέσα στη συμβολοσειρά.
- `startswith(substring)` → Επιστέφει `True` αν η συμβολοσειρά αρχίζει με την υπο-συμβολοσειρά *`substring`*.
- `endswith(substring)` → Επιστέφει `True` αν η συμβολοσειρά τελειώνει με την υπο-συμβολοσειρά *`substring`*.
- `replace(old, new)` → Επιστέφει ένα αντίγραφο της συμβολοσειράς στο οποίο όλες οι εμφανίσεις της υπο-συμβολοσειράς *`old`* έχουν αντικατασταθεί από την υπο-συμβολοσειρά *`new`*.

Άλλα χαρακτηριστικά των συμβολοσειρών

Ο τελεστής επανάληψης `*` λειτουργεί στις συμβολοσειρές όπως ακριβώς και στις λίστες, δηλαδή μπορεί να χρησιμοποιηθεί για να επαναλάβει μια συμβολοσειρά κάποιον συγκεκριμένο αριθμό φορών, ως εξής:

```
συμβολοσειρά * πλήθος_αντιγράφων
```

Π.χ.,

```
>>> my_str1 = 'a' * 10
>>> my_str1
'aaaaaaaaaa'
>>> my_str2 = 'Hello'*3
>>> my_str2
'HelloHelloHello'
```

Μια συμβολοσειρά μπορεί να διαχωριστεί στις λέξεις που την απαρτίζουν με τη χρήση της μεθόδου `split`. Η μέθοδος αυτή επιστρέφει μια ***λίστα*** με στοιχεία τις λέξεις που απαρτίζουν τη συμβολοσειρά με την οποία καλείται. Εξ ορισμού, η `split` χρησιμοποιεί ως διαχωριστικά λέξεων τα κενά. Μπορεί όμως να οριστεί και διαφορετικός χαρακτήρας διαχωρισμού, στέλνοντας τον επιθυμητό χαρακτήρα ως όρισμα στη μέθοδο. Π.χ.,

```
>>> my_str = 'Hello there!'
>>> words = my_str.split()
>>> words
['Hello', 'there!']
>>> date_str = '17/12/2015'
>>> date = date_str.split('/')
>>> date
['17', '12', '2015']
```