ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ
σε Python

Διάλεξη 5:
Πλειάδες και Συμβολοσειρές

Μάρκος Ζάμπογλου, mzampoglou@aegean.gr

Έννοιες ως τώρα (υπενθύμιση)

- **Λίστα (List)**: δυναμικό αντικείμενο που περιέχει αριθμημένα στοιχεία κάθε τύπου
  - **Indexing & Slicing**: Forward & backward, zero-indexed
  - Η εντολή **`in`**: Είτε λογική πράξη (αναζήτηση) είτε επανάληψη/iteration σε μια **`for`**
  - **Iterable**: μπορούμε να πάρουμε τα στοιχεία του διαδοχικά, π.χ. με μία **`for`**
  - **Mutable**: μπορούμε να τροποποιούμε τα στοιχεία του χωρίς να αλλάζουμε τις αναφορές σε αυτό
- **Συνάρτηση (Function)**: κώδικας με όνομα, εισόδους (*παραμέτρους*), εξόδους
- **Μέθοδος**: συνάρτηση ενσωματωμένη σε ένα αντικείμενο
- **Αντικείμενο**: στιγμιότυπο μιας κλάσης/τύπου
  - Μέθοδος που λειτουργεί **in-place**: τροποποιεί τις τιμές του αντικειμένου χωρίς απαραίτητα να έχει εξόδους
    - Π.χ. **`List.sort()`**, **`List.append()`**
- **Λίστα με λίστες:** αναφερόμαστε στα στοιχεία της με **`A[i][j][k]`**…
- **Σύνθεση** (**comprehension**): Ειδική σύνταξη που παράγει **λίστες** από **iterables**
- Στην Python όλες οι αναθέσεις είναι **by reference.** Αν τροποποιήσουμε μέρος ενός immutable, τροποποιούνται όλες οι μεταβλητές που δείχνουν σε αυτό.

Αντιγραφή δεδομένων λίστας

Υπάρχει ένας τρόπος να αντιγράφουμε τα περιεχόμενα μιας λίστας χωρίς να διατηρούμε την αναφορά στη θέση μνήμης της

```
list1 = [1,2,3,4]
list2 = list1.copy()
list2[1] = 9
print(list1)
print(list2)
[1, 2, 3, 4]
[1, 9, 3, 4]
```

Η μέθοδος **List.copy()** επιστρέφει ένα αντίγραφο της λίστας σε μια διαφορετική θέση μνήμης.

Οπότε όταν την αναθέτουμε σε μια μεταβλητή, αυτή η μεταβλητή είναι ανεξάρτητη από την αρχική λίστα.

Τα όρια της `list.copy()`

- Η δημιουργεί μια νέα λίστα που περιέχει όλα τα στοιχεία της αρχικής
  - Τι γίνεται αν ένα από αυτά τα στοιχεία είναι λίστα;

```
list1 = [1,2,[8,7],4]
list2 = list1.copy()
list2[2][0] = 2
print(list1)
print(list2)
```

```
[1, 2, [2, 7], 4]
[1, 2, [2, 7], 4]
```

Η **`list.copy()`** αντιγράφει τα δεδομένα μόνο στο πρώτο επίπεδο.

Αν υπάρχει «βάθος» (δηλ. λίστες μέσα σε λίστες), οι αναφορές διατηρούνται!

Βαθιά λύση: βιβλιοθήκη copy

```
import copy
list1 = [1,2,[8,7],4]
list2 = copy.deepcopy(list1)
list2[2][0] = 2
print(list1)
print(list2)
```

```
[1, 2, [8, 7], 4]
[1, 2, [2, 7], 4]
```

- Η μέθοδος **`copy.deepcopy()`** δημιουργεί αντίγραφα σε όλα τα πιθανά βάθη

Πλειάδες (Tuples)

- Οι πλειάδες είναι iterable ακολουθίες (sequence) όπως οι λίστες, όμως είναι immutable (αντί για mutable)
  - Δεν μπορούμε να τροποποιήσουμε κανένα επιμέρους στοιχείο τους
- Αντί για αγκύλες `[]`, ορίζονται με παρενθέσεις `()`

```
empty = ()
a = (10, 43, 423)
c = ("apple", 10, True, None, [3,4])
d = 43, 42, 66, 34, "Test", "True"
print(d)
(43, 42, 66, 34, 'Test', 'True')
print(d[3])
34
```

Indexing και slicing όμως κάνουμε κι εδώ με αγκύλες όπως στις λίστες!

Λειτουργίες πλειάδας

- Ως iterable και sequence μια πλειάδα έχει πολλές από τις δυνατότητες μιας λίστας:
  - ✓ Indexing, slicing
  - ✓ Πρόσθεση/συνένωση (`+`), πολλαπλασιασμό (`*`)
  - ✓ `in` ως λογική πράξη (π.χ. σε `if` ή `while`)
  - ✓ `in` για επανάληψη με `for`
  - ✓ Συναρτήσεις (`len()`, `min()`, `max()`, `sorted()`)
  - ✓ Μεθόδους που επιστρέφουν τιμές (`tuple.index()`, `tuple.count()`)
- Ως immutable όμως, **δεν επιτρέπει** εσωτερική τροποποίηση:
  - ✗ Απευθείας τροποποίηση τιμής: `a[3]=4`
  - ✗ Μεθόδους που εφαρμόζονται in-place (`.append()`, `.sort()`)
  - ✗ Comprehension
  - `del` για συγκεκριμένα στοιχεία: `del a[3]`
    - ✓ Επιτρέπεται φυσικά συνολικό `del a`

Πολλαπλή ανάθεση

Προσέξτε ότι μια σειρά αντικειμένων χωρισμένων με κόμματα δημιουργούν πλειάδα:

```
a = 10, 43, 23
print(d)
(10, 43, 23)
```

Άρα όταν κάνουμε πολλαπλή ανάθεση...

```
a, b = 10, 43
```

...στην πραγματικότητα κάνουμε ανάθεση πλειάδας σε πλειάδα

```
(a, b) = (10, 43)
```

Μπορούμε να κάνουμε το ίδιο και με λίστες

```
(a,b)=[3,4]
[a,b]=[3,4]
```

Η ενέργεια αυτή ονομάζεται **unpacking**

Πολλαπλή ανάθεση (unpacking)

```
a, b = 4, 7                 #a: 4, b: 7
c, *d = 6, 2, 4, 8          #c: 6, d: [2, 4, 8]
print(type(c),type(d))      #c: int d: list

# αλλά και επίσης
a, b = [4, 7]                   #a: 4, b: 7
a, b = (4, 7)                   #a: 4, b: 7
c, *d = [6, 2, 4, 8]            #c: 6, d: [2, 4, 8]
c, *d = (6, 2, 4, 8)            #c: 6, d: [2, 4, 8]
```

Γιατί υπάρχουν τα tuples;

- Οι πλειάδες επιτελούν τον ίδιο ρόλο με τις λίστες, όμως είναι immutable
  - Αυτό τις καθιστά **ακατάλληλες** για περιπτώσεις όπου θέλουμε να τις μεταβάλουμε
- Από την άλλη είναι πολύ πιο γρήγορες και οικονομικές σε μνήμη
  - Για αυτό είναι η default δομή για unpacking (π.χ. σε συναρτήσεις που έχουν πολλαπλές εξόδους)

Έννοιες (τελειώσαμε)

- **Λίστα (List)**: δυναμικό αντικείμενο που περιέχει αριθμημένα στοιχεία κάθε τύπου
  - **Indexing & Slicing**: Forward & backward, zero-indexed
  - Η εντολή **`in`**: Είτε λογική πράξη (αναζήτηση) είτε επανάληψη/iteration σε μια **`for`**
  - **Iterable**: μπορούμε να πάρουμε τα στοιχεία του διαδοχικά, π.χ. με μία **`for`**
  - **Mutable**: μπορούμε να τροποποιούμε τα στοιχεία του χωρίς να αλλάζουμε τις αναφορές σε αυτό
- **Συνάρτηση (Function)**: κώδικας με όνομα, εισόδους (*παραμέτρους*), εξόδους
- **Μέθοδος**: συνάρτηση ενσωματωμένη σε ένα αντικείμενο
- **Αντικείμενο**: στιγμιότυπο μιας κλάσης/τύπου
  - Μέθοδος που λειτουργεί **in-place**: τροποποιεί τις τιμές του αντικειμένου χωρίς απαραίτητα να έχει εξόδους
    - Π.χ. **`List.sort()`**, **`List.append()`**
- **Λίστα με λίστες:** αναφερόμαστε στα στοιχεία της με **`A[i][j][k]`**…
- **Σύνθεση** (**comprehension**): Ειδική σύνταξη που παράγει **λίστες** από **iterables**
- Στην Python όλες οι αναθέσεις είναι **by reference.** Αν τροποποιήσουμε μέρος ενός immutable, τροποποιούνται όλες οι μεταβλητές που δείχνουν σε αυτό.
- **Πλειάδα** (**Tuple**): ένα αντικείμενο που είναι **iterable** και **immutable**

Συμβολοσειρές

- Μια συμβολοσειρά (“string”) είναι μια ακολουθία από χαρακτήρες (γράμματα, αριθμητικά ψηφία, σύμβολα)
  - Τύπος: **`str`**
- Οι συμβολοσειρές είναι *sequences*, *iterable* και *immutable*

Ορισμός string

```
string0  =  ""
string0b =  str()
string1  =  "Hello"
string2  =  'Καλημέρα'
string3  =  '''Μπορούμε να έχουμε ένα
               string πολλών γραμμών
               αν το ορίσουμε με τριπλά
               εισαγωγικά'''
```

Κλείνουμε με ό,τι ανοίξαμε

Μονά ή διπλά εισαγωγικά

Εισαγωγικά μέσα σε string

Αν θέλουμε να γράψουμε κείμενο που περιέχει το σύμβολο των εισαγωγικών, επιτρέπεται να χρησιμοποιήσουμε μονά εισαγωγικά όταν ορίζουμε το string χρησιμοποιώντας διπλά -ή αντίστροφα:

```
a="Και γυρνάω και του λέω, 'Ξέρεις ποιός είμαι εγώ;'"
```

Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε το σύμβολο των ειδικών χαρακτήρων, "\"

```
b="Και γυρνάω και του λέω, \"Ξέρεις ποιός είμαι εγώ;\""
```

Μορφοποιημένα string

```
a = 3
print(f"Μορφοποιημένη εμφάνιση του αριθμού {a+2} εντός ενός string")
```
```
Μορφοποιημένη εμφάνιση του αριθμού 5 εντός ενός string
```

```
print(f"Μορφοποίηση με 2 υποχρεωτικά δεκαδικά ψηφία: 10/4={10/3:.2f}")
```
```
Μορφοποίηση με 2 δεκαδικά ψηφία: 10/3=3.33
```
```
print(f"Μορφοποίηση ως ποσοστό: 10/4={10/4:.3%}")
```
```
Μορφοποίηση με ως ποσοστό: 10/4=250.000%
```

Λειτουργίες συμβολοσειρών

- Iterable sequence όπως οι λίστες και τα tuples, οπότε επιτρέπουν:
  - ✓ Indexing, slicing
  - ✓ Πρόσθεση/συνένωση (`+`), πολλαπλασιασμό (`*`)
  - ✓ `in` ως λογική πράξη (π.χ. σε `if` ή `while`)
  - ✓ `in` για επανάληψη με `for`
  - ✓ Συναρτήσεις (`len()`, `min()`, `max()`, `sorted()`)
  - ✓ Μεθόδους που επιστρέφουν τιμές (`str.index()`, `str.count()`)
- Immutable όπως τα tuples, οπότε δεν επιτρέπουν:
  - ✗ Απευθείας τροποποίηση τιμής: `a[3]="κ"`
  - ✗ Μεθόδους που εφαρμόζονται in-place (`.append()`, `.sort()`)
  - ✗ Comprehension
  - ✗ `del` για συγκεκριμένα στοιχεία: `del a[3]`
    - ✓ Επιτρέπεται φυσικά συνολικό `del a`

Η εντολή in για string

```
word = "Όχι"
for c in word:
  print(c)
Ό
χ
ι
print("o" in word)
False
print("Ό" in word)
True
```

Συναρτήσεις σε string

```
string2='''Καλημέρα
 σας'''
print(len(string2))
13
print(max(string2))
σ
print(sorted(string2))
['\n', ' ', 'Κ', 'έ', 'α', 'α', 'α',
 'η', 'λ', 'μ', 'ρ', 'ς', 'σ']
```

Ο ειδικός χαρακτήρας **\n** σημαίνει «αλλαγή γραμμής». Ένας άλλος ενδιαφέρων ειδικός χαρακτήρας είναι ο **\t** που σημαίνει «Tab↹».

Συστήματα κωδικοποίησης χαρακτήρων

- Στους υπολογιστές, όλοι οι χαρακτήρες κειμένου κωδικοποιούνται με έναν μοναδικό αριθμό
- Η αντιστοίχιση χαρακτήρα-αριθμού καθορίζεται από ένα σύστημα κωδικοποίησης (*encoding*)
- Σήμερα η πιο διαδεδομένη κωδικοποίηση είναι η Unicode και συγκεκριμένα η **UTF-8**, που αποτελεί μετεξέλιξη της **ASCII**

https://en.wikipedia.org/wiki/List_of_Unicode_characters

Unicode example

Latin Alphabet: Uppercase	U+004B	K	75	0113	Latin Capital letter K	0044
	U+004C	L	76	0114	Latin Capital letter L	0045
	U+004D	M	77	0115	Latin Capital letter M	0046
	U+004E	N	78	0116	Latin Capital letter N	0047
	U+004F	O	79	0117	Latin Capital letter O	0048
	U+0050	P	80	0120	Latin Capital letter P	0049
	U+0051	Q	81	0121	Latin Capital letter Q	0050
	U+0052	R	82	0122	Latin Capital letter R	0051

Από χαρακτήρες σε κωδικούς

Επιστροφή του κωδικού ενός χαρακτήρα στο δεκαδικό:

```
print(ord('ή'))
```

```
942
```

Μετατροπή από δεκαδικό σε δεκαεξαδικό:

```
print(hex(ord('ή')))
```

```
0x3ae
```

Επιστροφή του χαρακτήρα από έναν δεκαδικό ή δεκαεξαδικό κωδικό:

```
print(chr(942))
```

```
ή
```

```
print(chr(0x3ae))
```

```
ή
```

Γενικές μέθοδοι

Τα string έχουν πρόσβαση στις ίδιες μεθόδους που έχουν π.χ. τα tuples

```
a = 'Hello'
# Πλήθος τιμών χαρακτήρα σε string
print(a.count('l'))
```

```
2
```

```
# Εντοπισμός θέσης χαρακτήρα σε string
print(a.index('o'))
```

```
4
```

```
print(a.index('x'))
```

```
Error
```

Παράδειγμα 5.1

Να γραφεί κώδικας που να δέχεται μια φράση στα αγγλικά και να επιστρέφει το πλήθος των φωνηέντων εντός της (`a`, `e`, `i`, `o`, και `u`). Υποθέστε ότι η φράση αποτελείται μόνο από μικρά γράμματα.

Λύση χωρίς comprehension:

Παράδειγμα 5.1

Να γραφεί κώδικας που να δέχεται μια φράση στα αγγλικά και να επιστρέφει το πλήθος των φωνηέντων εντός της (**a, e, i, o,** και **u**). Υποθέστε ότι η φράση αποτελείται μόνο από μικρά γράμματα.

Λύση χωρίς comprehension:

```
phrase = input("Δώσε μια φράση: ")

print(phrase_low.count('a') +
      phrase_low.count('e') +
      phrase_low.count('i') +
      phrase_low.count('o') +
      phrase_low.count('u'))
```

Μέθοδοι της κλάσης `str`

Όλες οι μέθοδοι για **`str`** επιστρέφουν μια τιμή. Δεν υπάρχουν συναρτήσεις για **`str`** που να λειτουργούν in-place.

Επιστρέφουν **`bool`** ή **`int`:**

- `str.find()`
- `str.isalpha()`
- `str.isprintable()`
- `str.islower()`
- `str.isupper()`
- `str.isnumeric()`
- `str.endswith()`
- `str.startswith()`

...και αρκετές ακόμη

Επιστρέφουν **`str`**:

- `str.upper()` και `str.lower()`
- `str.swapcase()`
- `str.strip()`
- `str.rstrip()`,`str.lstrip()`
- `str.replace()`
- `str.split()`
- `str.join()`

...και αρκετές ακόμη

Μέθοδοι που επιστρέφουν `bool` ή `int` (1/3)

`str.find()`: εξυπηρετεί τον ίδιο σκοπό με την **`index()`**, αλλά επιστρέφει -1 όταν η τιμή δεν υπάρχει:

```
print('Hello'.find('o'))
4
print('Hello'.find('x'))
-1
```

Σε όλα τα παραδείγματα οι μέθοδοι εφαρμόζονται πάνω σε σταθερά string (literals). Προφανώς στα προγράμματά μας ενδιαφέρει να τα εφαρμόσουμε πάνω σε μεταβλητές!

`str.isalpha()`: **`True`** αν το string αποτελείται αποκλειστικά από γράμματα:

```
print('abcfe'.isalpha())
True
print('12wv4!'.isalpha())
False
```

`str.isnumeric()`: **`True`** αν το string αποτελείται αποκλειστικά από αριθμούς:

```
print('1352'.isnumeric())
True
```

Μέθοδοι που επιστρέφουν `bool` ή `int` (2/3)

`str.isupper()`: **`True`** αν το string αποτελείται αποκλειστικά από κεφαλαία γράμματα:

```
print('Καλησπέρα'.isupper())
False
print('ΚΑΛΗΣΠΕΡΑ'.isupper())
True
```

`str.islower()`: **`True`** αν το string αποτελείται αποκλειστικά από μικρά γράμματα:

```
print('Καλησπέρα'.islower())
False
print('καλησπέρα'.islower())
True
```

`str.isprintable()`: **`True`** αν το string αποτελείται μόνο από χαρακτήρες που εμφανίζονται

```
print('Γραμμή 1\nΓραμμή 2'.islower())
False
```

Μέθοδοι που επιστρέφουν `bool` ή `int` (3/3)

`str.endswith(x)`: **`True`** αν το string τελειώνει με το string **`x`**:

```
print('Καλημέρα'.endswith('μέρα'))
True
print('Καλησπέρα'.endswith('μέρα'))
False
```

`str.startswith(x)`: **`True`** αν το string αρχίζει με το string **`x`**:

```
print('Καλημέρα'.startswith('Καλή'))
True
print('Καλημέρα'.startswith('καλή'))
False
```

Μέθοδοι που επιστρέφουν `str` (1/2)

`str.upper()`, **`str.lower()`**: μετατρέπει όλα τα γράμματα του string σε κεφαλαία και μικρά αντίστοιχα:

```
print('Καλημέρα'.upper())
ΚΑΛΗΜΕΡΑ
print('Καλημέρα'.lower())
καλημέρα
```

`str.swapcase()`: Αντιστρέφει κεφαλαία και μικρά:

```
print('Καλημέρα'.swapcase())
κΑΛΗΜΕΡΑ
```

`str.strip()`: Αφαιρεί κενά στην αρχή και στο τέλος του string:

```
print('   I love the smell of napalm in the morning    '.strip())
I love the smell of napalm in the morning
```

`str.lstrip()`, **`str.rstrip()`**: Αφαιρεί κενά μόνο από την αρχή ή μόνο από το τέλος του string:

Μέθοδοι που επιστρέφουν `str` (2/2)

`str.replace(old,new)`: αντικαθιστά όλες τις εμφανίσεις του string **`old`** με το string **`new`**:

```
print('Μου αρέσει η σουπίτσα'.replace('σ','θ'))
```

```
Μου αρέθει η θουπίτθα
```

```
print('Εφαγα ανανα και μπανανα'.replace('ανανα','μήλο'))
```

```
Εφαγα μήλο και μπμήλο
```

`str.split(sep)`: Διαχωρίζει το string σε όλες τις εμφανίσεις του διαχωριστή **`sep`** και το επιστρέφει ως λίστα:

```
print('Πώς να κόψεις μια πρόταση σε λέξεις;'.split(" "))
```

```
['Πώς', 'να', 'κόψεις', 'μια', 'πρόταση', 'σε', 'λέξεις;']
```

`str.join(L)`: Δέχεται μια λίστα **`L`** που αποτελείται από string και την επιστρέφει ενωμένη με το string της μεθόδου:

```
print('-'.join(['Λέξεις', 'σε', 'μια', 'λίστα']))
```

```
Λέξεις-σε-μια-λίστα
```

Παράδειγμα 5.2

Να γραφεί κώδικας σε Python που να δέχεται ένα string στα ελληνικά και να απαντάει αν πρόκειται για «παλίνδρομη» ή «καρκινική» φράση.

Καρκινική λέγεται μια φράση που διαβάζεται το ίδιο και από δεξιά προς αριστερά, π.χ. η λέξη «**Άννα**» ή η χριστιανική προτροπή «**ΝίψονΑνομήματαΜηΜόναν'Όψιν**».

- Ο κώδικας θα πρέπει να αγνοεί διαφορές μεταξύ μικρών και κεφαλαίων.
- Ο κώδικας θα πρέπει να αγνοεί τυχόν κενά (δηλαδή να αντιλαμβάνεται ως καρκινική και τη «**Νίψον Ανομήματα Μη Μόναν Όψιν**».
- Ο κώδικας θα πρέπει να θεωρεί τους τονισμένους χαρακτήρες ίδιους με τους άτονους, π.χ. το **ά** με το **α**.

Παράδειγμα 5.2

Να γραφεί κώδικας σε Python που να δέχεται ένα string στα ελληνικά και να απαντάει αν πρόκειται για «παλίνδρομη» ή «καρκινική» φράση.

Καρκινική λέγεται μια φράση που διαβάζεται το ίδιο και από δεξιά προς αριστερά, π.χ. η λέξη «**Άννα**» ή η χριστιανική προτροπή «**ΝίψονΑνομήματαΜηΜόναν Όψιν**».

- Ο κώδικας θα πρέπει να αγνοεί διαφορές μεταξύ μικρών και κεφαλαίων.
- Ο κώδικας θα πρέπει να αγνοεί τυχόν κενά (δηλαδή να αντιλαμβάνεται ως καρκινική και τη «**Νίψον Ανομήματα Μη Μόναν Όψιν**».
- Ο κώδικας θα πρέπει να θεωρεί τους τονισμένους χαρακτήρες ίδιους με τους άτονους, π.χ. το **ά** με το **α**.

```
test_str = input("Δώσε μια φράση: ")
# κεφαλαία σε μικρά
low = test_str.lower()
# αφαίρεση κενών
nospaces = low.replace(" ","")
# αντικατάσταση τόνων -θα μπορούσαμε
να συμπεριλάβουμε και διαλυτικά
noaccents = nospaces.replace("ά","α")
noaccents = noaccents.replace("έ","ε")
noaccents = noaccents.replace("ή","η")
noaccents = noaccents.replace("ί","ι")
noaccents = noaccents.replace("ό","ο")
noaccents = noaccents.replace("ύ","υ")
noaccents = noaccents.replace("ώ","ω")

print(noaccents == noaccents[::-1])
```

Παράδειγμα 5.2

Να γραφεί κώδικας σε Python που να δέχεται ένα string στα ελληνικά και να απαντάει αν πρόκειται για «παλίνδρομη» ή «καρκινική» φράση.

Καρκινική λέγεται μια φράση που διαβάζεται το ίδιο και από δεξιά προς αριστερά, π.χ. η λέξη «**Άννα**» ή η χριστιανική προτροπή «**ΝίψονΑνομήματαΜηΜόναν'Όψιν**».

- Ο κώδικας θα πρέπει να αγνοεί διαφορές μεταξύ μικρών και κεφαλαίων.
- Ο κώδικας θα πρέπει να αγνοεί τυχόν κενά (δηλαδή να αντιλαμβάνεται ως καρκινική και τη «**Νίψον Ανομήματα Μη Μόναν Όψιν**».
- Ο κώδικας θα πρέπει να θεωρεί τους τονισμένους χαρακτήρες ίδιους με τους άτονους, π.χ. το **ά** με το **α**.

```
test_str = input("Δώσε μια φράση: ").lower().replace(" ","").replace("ά","α")
    .replace("έ","ε").replace("ή","η").replace("ί","ι").replace("ό","ο")
    .replace("ύ","υ").replace("ώ","ω")
```

Παράδειγμα 5.3

Να γραφεί κώδικας σε Python που θα ελέγχει αν μια διεύθυνση email είναι ορθή με βάση τους παρακάτω κανόνες:

- Κανένα κενό
- Μια εμφάνιση του συμβόλου @
- Το domain (το τμήμα δεξιά του @) πρέπει να έχει τουλάχιστον μια τελεία
- Το local part (το τμήμα αριστερά του @) πρέπει να έχει τουλάχιστον έναν χαρακτήρα

Παράδειγμα 5.3

Να γραφεί κώδικας σε Python που θα ελέγχει αν μια διεύθυνση email είναι ορθή με βάση τους παρακάτω κανόνες:

- Κανένα κενό
- Μια εμφάνιση του συμβόλου @
- Το domain (το τμήμα δεξιά του @) πρέπει να έχει τουλάχιστον μια τελεία
- Το local part (το τμήμα αριστερά του @) πρέπει να έχει τουλάχιστον έναν χαρακτήρα

```
email = input("Δώσε τη διεύθυνσή σου:")

if email.count(" ")>0:
  print("Τα κενά απαγορεύονται")
elif email.count("@")!=1:
  print("Πρέπει να υπάρχει ένα (και μόνο ένα) παπάκι")
else:
  [localpart,domain]=email.split("@")
  if domain.count(".")==0:
    print("Το domain δεν έχει '.'")
  elif len(localpart)==0:
    print("κενό localpart")
  else:
    print("Επιτρεπτή διεύθυνση")
```

List comprehension σε string

- Είπαμε ότι δεν υπάρχει string comprehension
- Μπορούμε όμως να κάνουμε list comprehension διατρέχοντας τα στοιχεία ενός string
  - Όπως οποιουδήποτε άλλου iterable
- Είναι ένας αποτελεσματικός τρόπος να σαρώνουμε τα στοιχεία ενός string

```
text = "5 gold rings, 4 calling birds, 3 French
        hens, 2 turtle doves, and a partridge in
        a pear tree!"
digits = [c for c in text if c.isnumeric()]
print(digits)
['5', '4', '3', '2']
```

Παράδειγμα 5.4

Να γραφεί κώδικας σε Python που θα δέχεται έναν κωδικό και θα επιστρέφει αν είναι ισχυρός ή όχι. Για να είναι ισχυρός θα πρέπει να περιέχει τουλάχιστον 8 χαρακτήρες, εκ των οποίων ένας θα είναι ένα κεφαλαίο γράμμα και ένας θα είναι αριθμός.

- *Κι αν πρέπει να περιέχει και ένα από τα σύμβολα [!,%,&,$,#,*];*

Παράδειγμα 5.4

Να γραφεί κώδικας σε Python που θα δέχεται έναν κωδικό και θα επιστρέφει αν είναι ισχυρός ή όχι. Για να είναι ισχυρός θα πρέπει να περιέχει τουλάχιστον 8 χαρακτήρες, εκ των οποίων ένας θα είναι ένα κεφαλαίο γράμμα και ένας θα είναι αριθμός.

- *Κι αν πρέπει να περιέχει και ένα από τα σύμβολα [!,%,&,$,#,*];*

```
passw = input("Δώσε κωδικό: ")

mikos_ok = len(passw)>8
caps = [c for c in passw if c.isupper()]
caps_ok = len(caps)>0
numbers = [c for c in passw if c.isnumeric()]
numbers_ok = len(numbers)>0
symbols =  [c for c in passw if c in ['!','%','&','$','#','*']]
symbols_ok = len(symbols)>0

if not mikos_ok:
  print("Πρέπει να είναι τουλάχιστον 8 χαρακτήρες.")
elif not caps_ok:
  print("Πρέπει να περιέχει τουλάχιστον ένα κεφαλαίο γράμμα.")
elif not numbers_ok:
  print("Πρέπει να περιέχει τουλάχιστον έναν αριθμό.")
elif not symbols_ok:
  print("Πρέπει να περιέχει τουλάχιστον ένα σύμβολο.")
else:
  print("Αποδεκτός κωδικός")
```

Παράδειγμα 5.5

Να γραφεί κώδικας που να δέχεται μια φράση στα αγγλικά και να επιστρέφει το πλήθος των φωνηέντων εντός της (`a`, `e`, `i`, `o`, και `u`)

Λύση με comprehension:

Παράδειγμα 5.5

Να γραφεί κώδικας που να δέχεται μια φράση στα αγγλικά και να επιστρέφει το πλήθος των φωνηέντων εντός της (**a, e, i, o,** και **u**)

Λύση με comprehension:

```
phrase = input("Give a phrase: ")
vowel_list = 'aeiou'

all_vowels = [c for c in phrase if c in vowel_list]
print(len(all_vowels))
```