ΠΕΡΙΕΧΟΜΕΝΑ:

Νικόλαος Λουκάς

Ασημένιος Χορηγός Μαθήματος

Μιχάλης Γ.

Χάλκινος Χορηγός Μαθήματος

- **Πλειάδα (tuple)** είναι μία συλλογή (collection) με μία συγκεκριμένη σειρά. **Οι τιμές δεν μπορούν να αλλάξουν (immutable)**
  - Προσοχή, ότι υπάρχει **διάταξη** (εκτός από το ποια αντικείμενα αποθηκεύονται, μας ενδιαφέρει και η σειρά με την οποία αυτά αποθηκεύονται.
  - Έχει παρόμοια συμπεριφορά με τη λίστα, με τη διαφορά ότι στη λίστα μπορούμε να τροποποιήσουμε τις τιμές (**mutable**)

- **Δήλωση πλειάδας**
  - Ορίζουμε μια πλειάδα χρησιμοποιώντας κάποιο όνομα μεταβλητής και ενθέτοντας τις τιμές σε παρενθέσεις.

Παράδειγμα 1: tuples.py

Στο παράδειγμα βλέπουμε τον ορισμό 4 πλειάδων:

```
int_tuple = (1, 2, 3)
collection = ("hi", 3.14, True)
tuple_of_one = (1,)
empty_tuple = ()

print(collection)
print(type(tuple_of_one))
```

Προσέχουμε την ιδιαιτερότητα του tuple με ένα στοιχείο (το κόμμα είναι απαραίτητο).

- **Συμπεριφορά:**
  - Ίδια με αυτή της λίστας:
    - indexing: π.χ. my_tuple[1], my_tuple[-2]
    - Εύρη: π.χ. my_tuple[0:4]
    - έλεγχοι: π.χ. if element in tuple:
    - loops: π.χ. for element in tuple:
  - Προφανώς, δεν έχει μεθόδους που τροποποιούν την πλειάδα, όπως π.χ. η pop() και η append()

Χρησιμοποιούμε την πλειάδα:

- Αν ξέρουμε ότι τα δεδομένα του πίνακα δεν πρόκειται να αλλάξουν, προτιμάμε την πλειάδα από τη λίστα.

Παράδειγμα 2: tuples2.py

```
my_tuple = (1, 2, 3)

print(my_tuple[1:3])
print(my_tuple[1] + 4)
for number in my_tuple:
    print(number)
```

- **Η πλειάδα είναι και αυτή ακολουθία:**
  - Όπως και η λίστα, το range, το string και κάποια άλλα που θα μάθουμε σε επόμενα μαθήματα.
  - Οι ακολουθίες έχουν κοινή συμπεριφορά με συναρτήσεις και μεθόδους που εφαρμόζονται με ίδιο τρόπο σε κάθε τύπο ακολουθίας.

- **Κοινή συμπεριφορά όλων των ακολουθιών:**

Πράξη	Εξήγηση
elem in seq	T/F αν το elem ανήκει στην ακολουθία
elem not in seq	T/F αν το elem δεν ανήκει στην ακολουθία
seq1 + seq2	Συνένωση (για ομοειδείς ακολουθίες)
seq*n ή n*seq	Συνένωση n φορές της seq
seq[i], seq[i:j], seq[i,j,k]	Indexing, slicing και slicing με βήμα k
len(seq)	Μήκος της ακολουθίας
min(seq)	Ελάχιστο στοιχείο της ακολουθίας
max(seq)	Μέγιστο στοιχείο της ακολουθίας
seq.index(elem, s, f)	Επιστρέφει τη θέση της πρώτης εμφάνισης του elem στη seq (στο μέρος από το s στο f (προαιρετικά ορίσματα)
seq.count(elem)	Επιστρέφει το συνολικό πλήθος εμφανίσεων του elem στη seq

Χρήσιμες Μετατροπές

- Μετατροπή λίστας σε tuple:

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
```

- Μετατροπή tuple σε λίστα:

```
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
```

- Μετατροπή range σε λίστα:

```
my_list = list(range(4))
print(my_list)
```

- Μετατροπή string σε λίστα:

```
msg = "Hello!"
print(list(msg))
```

(βλ.και: conversions.py)

Παράδειγμα 2: sequence.ops.py

```
numbers = (1,2,3,4,3,2,3,3)
print("Length: " + str(len(numbers)))
print("min: " + str(min(numbers)))
print("max: " + str(max(numbers)))
print("3s: " + str(numbers.count(3)))
print("pos of a 3: " + str(numbers.index(3,4)))
```

Παράδειγμα 3: brain.damage.py

```
my_list = [1,2,3]
new_list = ((my_list * 2)[1:5] + list((7,8)))*4
print(str((my_list+new_list).count(2)))
```

Παράδειγμα 4: list.copy.problem.py

Δείτε ότι το παρακάτω πρόγραμμα δεν έχει την επιθυμητή συμπεριφορά:

```
list1 = [1,2,3]
list2 = list1
list2[0] = 4
print(list1)
print(list2)
```

Αντιγραφή Ακολουθιών:

- Μεγάλη προσοχή όταν θέλουμε να αντιγράψουμε μια ακολουθία σε μια άλλη (λίστα ή tuple).
  - Είτε θα χρησιμοποιήσουμε slicing χωρίς όρια:

    ```
    new_list = old_list[:]
    ```

  - Είτε θα χρησιμοποιήσουμε τη μέθοδο copy()

    ```
    new_list = old_list.copy()
    ```

- Πιο γρήγορο είναι το slicing και θα το προτιμάμε.

Σημείωση:

- Το όνομα της λίστας και του tuple είναι μία αναφορά και όχι συνηθισμένη μεταβλητή.
- Οι αναφορές παρουσιάζουν τέτοια προβλήματα στην αντιγραφή.
- Θα το καταλάβουμε πλήρως σε επόμενο μάθημα.

Παράδειγμα 5: list.copy.py

```
list1 = [1,2,3]
list2 = list1[:]
list2[0] = 4
print(list1)
print(list2)
```

Άσκηση 1: Δυναμική κατασκευή tuples

Για κάποιους μυστήριους λόγους θέλουμε να κατασκευάσουμε ένα πρόγραμμα το οποίο:

- Θα διαβάζει από την είσοδο 10 ακεραίους μεταξύ 10 και 20 και θα τους αποθηκεύει σε μία λίστα
- Θα δημιουργεί ένα tuple με περιεχόμενο αυτούς τους 10 αριθμούς.
- Θα δημιουργεί ένα δεύτερο tuple το οποίο θα περιέχει ταξινομημένα, τα τετράγωνα αυτών των 10 αριθμών.

Άσκηση 2: Δένδρο (ξανά)

Ξαναλύστε την άσκηση για την εκτύπωση του δένδρου, αξιοποιώντας τον τελεστή * πάνω σε συμβολοσειρές:
Υπενθύμιση για N=5:

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
```

Άσκηση 3: Πρώτος Αριθμός

Κατασκευάστε ένα πρόγραμμα το οποίο:

- Θα ορίζει ένα μη αρνητικό ακέραιο αριθμό N
- Θα ελέγχει αν είναι πρώτος και θα τυπώνει το αποτέλεσμα

Υπενθύμιση: Τα 0 και 1 δεν είναι πρώτοι και κάθε φυσικός >= 2 είναι πρώτος ανν διαιρείται (ακριβώς) μόνο με τον εαυτό του και τη μονάδα.

Άσκηση 4: Πρώτοι αριθμοί

Κατασκευάστε ένα πρόγραμμα το οποίο:

- Θα κατασκευάζει και θα τυπώνει ένα tuple με τους πρώτους αριθμούς που υπάρχουν από το 2 έως 100.

- **Οι περιγραφικές λίστες (list comprehensions):**
  - Παρέχουν ένα "πιο γρήγορο" συντακτικό για να κατασκευάσουμε επαναληπτικά μια λίστα

- **Περίπτωση 1: Μια απλή for..in**
- Αντί να γράψουμε

```
my_list = []
for number in range(3):
    my_list.append(number)
```

- μπορούμε να γράψουμε περιγραφικά:

```
my_list = [number for number in range(3)]
```

(βλ. και comprehensions.py)

- **Περίπτωση 2: Μια for…in με if**
- Αντί να γράψουμε

```
my_list = []
for number in range(10):
    if number % 2 == 0:
        my_list.append(number)
```

- μπορούμε να γράψουμε:

```
my_list = [number for number in range(10) if number%2 == 0]
```

(βλ. και comprehensions2.py)

- **Περίπτωση 3: Μια for…in με if…else**
- Αντί να γράψουμε

```
my_list = []
for number in range(10):
    if number % 2 == 0:
        my_list.append(number)
    else:
        my_list.append(number/2)
```

- μπορούμε να γράψουμε (σε μία γραμμή) το παρακάτω:

```
my_list = [number if number%2 == 0 else number/2
                    for number in range(10)]
```

(βλ. και comprehensions3.py)

Άσκηση 5:

Κατασκευάστε μία λίστα που περιέχει τους άρτιους αριθμούς που είναι και πολλαπλάσια του 3, χρησιμοποιώντας για την κατασκευή, περιγραφική λίστα.

- **Μία διδιάστατη λίστα** ορίζεται απλά ως μία λίστα που περιέχει άλλες λίστες.
- π.χ. ορίζουμε έναν 2x3 πίνακα:

```
my_list2d = [
   [1, 2, 3],
   [4, 5, 6]
]
```

- ένα στοιχείο της λίστας (π.χ. my_list2d[0]) είναι μία λίστα
- και έχουμε πρόσβαση σε κάποιο στοιχείο με την πρόσβαση σε αυτό μέσο της λίστας-γραμμής, π.χ.: my_list2d[0][1]
- Κατά τα λοιπά, μπορούμε να εφαρμόσουμε τις πράξεις λιστών που μάθαμε στα μαθήματα 4 και 6.

Παράδειγμα 6: array2d.py

Κατασκευή ενός 2Δ πίνακα από την είσοδο του χρήστη:

```
array = []
rows = int(input("Give number of rows: "))
cols = int(input("Give number of columns: "))
for i in range(rows):
   array.append([])
   for j in range(cols):
      elem = int(input("Give " + str(i) + "," + str(j) + " element: "))
      array[i].append(elem)
print(array)
```

Παράδειγμα 6: array2d.traverse.py

Το πρόγραμμα αλλάζει την τιμή σε ένα στοιχείο και έπειτα διατρέχει τον πίνακα:

```
array = [[1,2,3], [4,5,6]]
array[0][1] = 0
for row in array:
   for elem in row:
      print(elem, end=" ")
   print("")
```

Σημείωση:

- Αντίστοιχα ορίζονται 3Δ πίνακες (ως μία λίστα που περιέχει 2Δ πίνακες)
- Δεν επεκτεινόμαστε στη μαθηματική χρήση, μιας και το πακέτο numPy, προσφέρει ολοκληρωμένη μαθηματική λειτουργικότητα

Άσκηση 6:

Κατασκευάστε μία λίστα που απεικονίζει έναν 3x4 πίνακα (με στοιχεία της επιλογής σας) και έπειτα:

- Προσθέτει μία γραμμή μηδενικών στην αρχή
- τυπώνει τον 4x4 πίνακα
- Προσθέτει μία στήλη με άσσους στο τέλος
- τυπώνει τον 4x5 πίνακα

Σχολιάζουμε τον κώδικά μας όταν:

- κάτι χρήζει επεξήγησης, για να γίνει πιο ευανάγνωστος, ή σαν σχόλιο κατά την κατασκευή του για να θυμόμαστε κάτι.

Σχόλια μίας γραμμής:

- Ξεκινά με #
- οτιδήποτε ακολουθεί τη # μέχρι το τέλος της γραμμής είναι σχόλιο.

```
# My brilliant code starts here
if rocket is ready:
    launch("mars") # change planet if necessary
```

Σχόλια πολλών γραμμών:

- Ξεκινά με """ (τριπλά διπλά εισαγωγικά)
- Γράφουμε τον εκτενή σχολιασμό μας στις επόμενες γραμμές
- Τελειώνει με """ (τριπλά διπλά εισαγωγικά)

```
"""
This code is not working.
read_python is not working properly..
"""
while kim not presses_button:
    read_python()
```

Όπως θα δούμε σε επόμενο μάθημα, αυτό είναι απλά ένα string πολλών γραμμών, που απλά δεν αποθηκεύεται κάπου.

Στυλ Κώδικα:

- Είναι κάποιες συμβάσεις για να είναι ο κώδικάς μας πιο ευανάγνωστος.
- Guido: «Πιο πολύ χρόνο αφιερώνουμε για να διαβάσουμε τον κώδικα μας, παρα για να τον γράψουμε»
- Το **PEP8** (Python Enhancement Protocol 8 – google it!) καθορίζει συστάσεις που πρέπει να ακολουθούν οι προγραμματιστές για το στυλ κώδικα.

Παραδείγματα:

- Η στοίχιση να γίνεται με 4 κενά και όχι με tabs.
- Όταν σπάει μία γραμμή σε πολλαπλές γραμμές να γίνεται η στοίχιση να γίνεται με 4 επιπλέον κενά.
- Κράτα τις γραμμές κώδικα μέχρι 79 χαρακτήρες (ώστε να χωράνε στην κονσόλα)
- Μία λίστα που απεικονίζεται σε πολλές γραμμές, έχει την αγκύλη που ανοίγει στην 1η γραμμή, μετά τα στοιχεία με ένα επίπεδο στοίχισης στις επόμενες και τέλος την αγκύλη που κλείνει στην τελευταία γραμμή χωρίς στοίχιση.
- Οι κλάσεις και οι ο ορισμός συναρτήσεων πρέπει να ακολουθούνται από δύο κενές γραμμές.

Το PyCharm:

- **Ενσωματώνει αυτές τις οδηγίες.**
- και υπογραμμίζει τα "λάθη" στο στυλ κώδικα.
- Μπορούμε να ακολουθούμε τις οδηγίες του για να ενσωματώνουμε στο πρόγραμμά μας αυτές τις συστάσεις.

Άσκηση 7.1: Παιχνίδι Μνήμης με Κάρτες

Στο γνωστό παιχνίδι μνήμης με κάρτες:

- Μας δίνεται μια σειρα από κλειστές κάρτες, ανακατεμένες
- Οι κάρτες είναι διάδες, δηλαδή ανά δύο έχουν το ίδιο σύμβολο.
- Ο παίκτης διαλέγει δύο κάρτες, τις ανοίγει και:
  - Αν είναι ίδιες, τις αφήνει ανοικτές
  - Αν είναι διαφορετικές, τις ξανακλείνει.

Ξεκινήστε την κατασκευή ορίζοντας μία λίστα, μήκους 8, που θα περιέχει 2 φορές τους αριθμούς 1,2,3,4

- Τοποθετήστε τους αριθμούς σε τυχαία σειρά
- (για την ώρα το ανακάτεμα των αριθμών το κάνουμε καρφωτά (hard-coded), αργότερα θα μάθουμε πως να το αρχικοποιούμε με τυχαίο τρόπο)

Άσκηση 7.2: Λίστα τρέχουσας κατάστασης

Ορίστε μια ξεχωριστή λίστα που δείχνει την τρέχουσα κατάσταση του παιχνιδιού. Επιλέξτε μια κωδικοποίηση των στοιχείων, ώστε να απεικονίζονται οι μαντεψιές του χρήστη:

- Να φαίνεται ποιες κάρτες είναι κλειστές.
- Να φαίνεται ποιες κάρτες είναι ανοικτές.
- Να φαίνεται ποιες κάρτες είναι προσωρινά ανοικτές.

Άσκηση 7.3: Μαντεψιές

Επεκτείνετε το πρόγραμμα έτσι ώστε ο παίκτης επαναληπτικά:

- Επιλέγει 2 θέσεις (πρέπει να είναι από το 0 έως το 7 και οι αντίστοιχες κάρτες να είναι κλειστές)
- Τυπωνεται η τρέχουσα κατάσταση με ανοικτές τις δύο κάρτες.
- Αν είναι ίδιες, μένουν ανοικτές, αλλιώς ξανακλείνουν.

Εως ότου ο ανοίξουν όλες οι κάρτες οπότε το παιχνίδι τελειώνει.

Δώστε λειτουργικότητα, με κατάλληλα μηνύματα προς τον παίκτη.

Άσκηση 7.4: Πλήθος προσπαθειών (σκορ)

Επεκτείνετε το πρόγραμμα ώστε να μετράει το πλήθος των προσπαθειών του παίκτη:

- Στο τέλος του παιχνιδιού να εμφανίζει πόσες προσπάθειες έκανε ο παίκτης.