

---

# 32. ΑΝΑΓΝΩΣΗ ΑΡΧΕΙΩΝ – LIST COMPREHENSION – LAMBDA FUNCTIONS - ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΕΙΑ

---

Επανάληψη κεφαλαίων 9-12

## 32.0.0 Λύσεις των προηγούμενων ασκήσεων

### Άσκηση 1 (guessNum1-20\_8.12)

'''

*Γράψτε ένα πρόγραμμα που επιλέγει έναν αριθμό  
μεταξύ 1 και 20 και ζητά από το χρήστη να  
μαντέψει ποιος είναι αυτός ο αριθμός.*

*Το πρόγραμμα θα πρέπει να επαναλαμβάνεται  
μέχρι ο χρήστης να μαντέψει σωστά.*

'''

```
import random

secret = random.randint(1,21)
guess = 0
while guess != secret:
    guess = int(input("Προσπαθήστε να βρείτε τον μυστικό  
αριθμό από το 1 μέχρι το 20: "))
    if guess < secret:
        print("Ξαναδοκιμάστε βάζοντας ένα μεγαλύτερο αριθμό:")
    "
```

```
elif guess > secret:
    print("Ξαναδοκιμάστε βάζοντας ένα μικρότερο αριθμό:")
print("Μπράβο! Το βρήκατε! Είναι ο αριθμός ",secret)
```

## Άσκηση 2 (factorial\_8.11)

'''

*Το factorial ενός αριθμού, είναι ο αριθμός των γινομένων όλων των προηγούμενων του, από το 1 μέχρι και τον ίδιο.*

*Π.χ. Το factorial του 3 (!3), είναι:  $1*2*3 = 6$*

*Γράψτε ένα πρόγραμμα που ζητά από το χρήστη να εισάγει έναν ακέραιο αριθμό και επιστρέφει το factorial του*

'''

```
n = int(input("Δώστε έναν ακέραιο αριθμό: "))
factorial = 1
for i in range(1, n+1):
    factorial *= i
# Μπορούμε να βάλουμε διαχωριστή χιλιάδων ","
print("Το factorial είναι{:,}" .format(factorial))
# βγάλτε το hash «#» για να τυπώσετε τη φράση
print("Το factorial του", n, "είναι", factorial)
```

## Άσκηση 3: Προσάρτηση σε ένα αρχείο

'''

*Ανοίξτε το αρχείο "my\_file.txt" και προσθέστε το παρακάτω κείμενο στο τέλος του αρχείου: "This is some new text added to the file."*

'''

```
with open("my_file.txt", "a") as f:
    f.write("This is some new text added to the file.")
```

## Άσκηση 4: Ανάγνωση ενός αρχείου γραμμή προς γραμμή

'''

Δημιουργήστε ένα αρχείο με το όνομα "my\_poem.txt" και γράψτε το παρακάτω ποίημα σε αυτό:

*Roses are red,*

*Violets are blue,*

*Sugar is sweet,*

*And so are you.*

Ανοίξτε το αρχείο "my\_poem.txt" και διαβάστε το περιεχόμενό του γραμμή προς γραμμή. Εκτυπώστε κάθε γραμμή στην κονσόλα. '''

```
with open("my_poem.txt", "r") as f:
    for line in f:
        print(line)
```

## Άσκηση 5: Δημιουργία αρχείου αν δεν υπάρχει

'''

Ανοίξτε το αρχείο "my\_notes.txt" αν υπάρχει ή δημιουργήστε ένα νέο αρχείο με αυτό το όνομα αν δεν υπάρχει. Γράψτε το παρακάτω κείμενο στο αρχείο: "These are my notes for Python."

'''

```
with open("my_notes.txt", "w") as f:
    f.write("These are my notes for Python.")
```

## Άσκηση 6: Προσάρτηση σε ένα αρχείο αν δεν υπάρχει

'''

Προσθέστε το παρακάτω κείμενο στο αρχείο "my\_notes.txt" αν υπάρχει ή δημιουργήστε ένα νέο αρχείο με αυτό το όνομα αν δεν υπάρχει:

*"These are some additional notes for Python."*

'''

```
with open("my_notes.txt", "a") as f:  
    f.write("These are some additional notes for Python.")
```

## Άσκηση 7: Αντιγραφή περιεχομένου από ένα αρχείο σε ένα άλλο

'''

*Δημιουργήστε ένα αρχείο με το όνομα "source.txt" και γράψτε το παρακάτω κείμενο σε αυτό: "This is the content of the source file."*

*Δημιουργήστε ένα κενό αρχείο με το όνομα "destination.txt".*

*Ανοίξτε τα δύο αρχεία και αντιγράψτε το περιεχόμενο του "source.txt" στο "destination.txt".*

'''

```
with open("source.txt", "r") as source,  
    open("destination.txt", "w") as  
    destination:  
    contents = source.read()  
    destination.write(contents)
```

## 32.1.0 Χρήση της `read()` για ανάγνωση αρχείων με το μέγεθος του buffer

Μερικές φορές, για να μην δεσμεύουμε πόρους της μνήμης, μπορεί να θέλουμε να διαβάσουμε ένα αρχείο με το μέγεθος ενός buffer. Για να το κάνουμε αυτό, χρησιμοποιούμε τη συνάρτηση `read()`, κι όχι τη `readline()`, με την οποία μπορούμε να καθορίσουμε το μέγεθος του buffer.

Παράδειγμα:

```
inputFile = open ('myfile.txt', 'r')
outputFile = open ('myoutputfile.txt', 'w')
msg = inputFile.read(10)
while len(msg):
    outputFile.write(msg)
    msg = inputFile.read(10)
inputFile.close()
outputFile.close()
```

Πρώτα, ανοίγουμε δύο αρχεία. Το **`inputFile.txt`** και το **`outputFile.txt`** για ανάγνωση και γραφή αντίστοιχα.

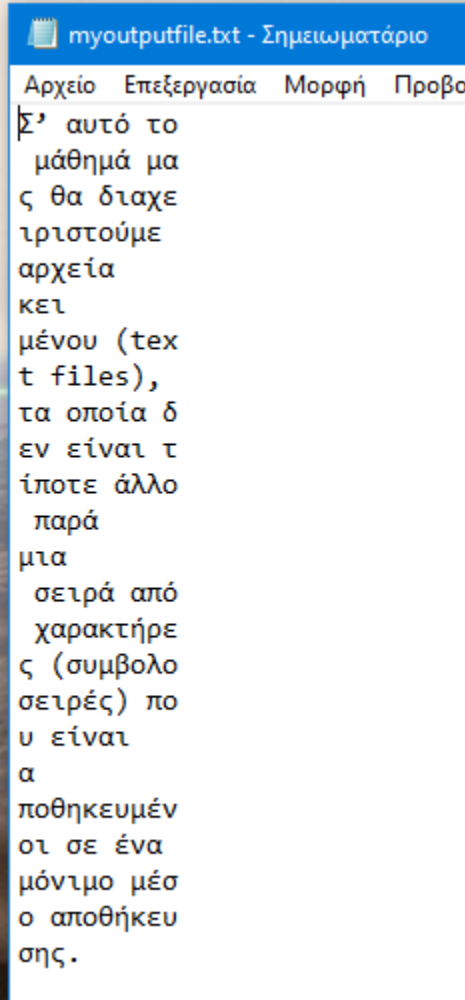
Κατόπιν, χρησιμοποιούμε τη δήλωση **`msg = inputFile.read(10)`** και ένα **`while loop`** για να διαβάσει το αρχείο ανά 10 bytes τη φορά. Αυτό το δηλώνουμε με την τιμή «10» στις παρενθέσεις της **`read()`**.

Η συνθήκη **`while, while len(msg)`**: ελέγχει το μήκος της μεταβλητής **`msg`**. όσο αυτό το μήκος δεν είναι ίσο με το μηδέν, ο βρόχος συνεχίζει να τρέχει.

Εντός του βρόχου `while`, η δήλωση **`outputFile.write(msg)`** γράφει το μήνυμα στο αρχείο εξόδου. Αφού γράψουμε το μήνυμα, η δήλωση **`msg = inputFile.read(10)`** διαβάζει τα επόμενα 10 byte και συνεχίζει να το κάνει μέχρι να διαβαστεί όλο το αρχείο. Όταν συμβεί αυτό, το πρόγραμμα κλείνει και τα δύο αρχεία.

Όταν εκτελούμε το πρόγραμμα, θα δημιουργηθεί ένα νέο αρχείο **`myoutputfile.txt`**. Όταν ανοίξουμε το αρχείο, θα παρατηρήσουμε ότι έχει το ίδιο περιεχόμενο με το αρχείο εισόδου μας **`myfile.txt`**. Για να δούμε αν διαβάζονται μόνο 10 byte κάθε φορά, μπορούμε να αλλάξουμε τη γραμμή **`outputFile.write(msg)`** στο πρόγραμμα σε **`outputFile.write(msg + '\n')`**.

Τώρα, αν εκτελέσουμε ξανά το πρόγραμμα, το **myoutputfile.txt** περιέχει γραμμές με το πολύ 10 χαρακτήρες. Ας δούμε το αρχιάκι μας τώρα:



```
1 ξ' αυτό το  
2 μάθημά μα  
3 ς θα διαχε  
4 ιριστούμε  
5 αρχεία  
6 κει  
7 μένου (tex  
8 t files),  
9 τα οποία δ  
10 εν είναι τ  
11 ίποτε άλλο  
12 παρά  
13 μια  
14 σειρά από  
15 χαρακτήρε  
16 ς (συμβολο  
17 σειρές) πο  
18 υ είναι  
19 α  
20 ποθηκευμέν  
οι σε ένα  
μόνιμο μέσ  
ο αποθήκευ  
σης.
```

### 32.1.1 Άνοιγμα, ανάγνωση κι εγγραφή δυαδικών αρχείων

Τα δυαδικά αρχεία αναφέρονται σε οποιοδήποτε αρχείο που δεν περιέχει κείμενο, όπως εικόνα ή βίντεο. Για να δουλέψουμε με δυαδικά αρχεία, χρησιμοποιούμε απλώς τη λειτουργία 'rb' ή 'wb'. Αντιγράφουμε ένα αρχείο jpg στην επιφάνεια εργασίας σας και το μετονομάζουμε σε *myimage.jpg*. Τώρα αλλάζουμε στο παραπάνω πρόγραμμα τις δύο πρώτες γραμμές:

```
inputFile = open ('myfile.txt', 'r')
outputFile = open ('myoutputfile.txt', 'w')
ΣΕ:
inputFile = open ('myimage.jpg', 'rb')
outputFile = open ('myoutputimage.jpg', 'wb')
try:
    imageData = inputFile.read()
    outputFile.write(imageData)
    print("Image opened and copied successfully.")
finally:
    inputFile.close()
    outputFile.close()
```

Εκτελώντας το νέο πρόγραμμα έχουμε ένα επιπλέον αρχείο εικόνας με όνομα *myoutputimage.jpg* στην επιφάνεια εργασίας μας. Όταν ανοίξουμε το αρχείο εικόνας, θα πρέπει να είναι ακριβώς το ίδιο με το *myimage.jpg*.



myoutputimage.jpg

## 32.2.0 List Comprehensions

Προγραμματίζοντας με την Python, γνωρίζουμε ότι από τη στιγμή που έχουμε μια λίστα, είναι πολύ πιθανό να χρειαστεί να γράψουμε ένα βρόχο. Τις περισσότερες φορές αυτό είναι εντάξει, αλλά η Python μας δίνει τη δυνατότητα, όταν πρόκειται για κάτι σχετικά απλό, να μπορούμε να χρησιμοποιήσουμε τη σύνταξη list comprehension, η οποία γράφεται πολύ πιο σύντομα, σε μια γραμμή.

Πρόκειται για λίστες οι οποίες αναπαράγουν τον εαυτό τους με έναν εσωτερικό βρόχο. Είναι κοινό χαρακτηριστικό στην Python και συνήθως μοιάζουν με:

```
x for x in list_of_x's
```

Όπως γνωρίζουμε, οι λίστες έχουν πολλά αντικείμενα και καθορίζονται με τις αγκύλες. Ας δημιουργήσουμε ένα παράδειγμα, φτιάχνοντας μια



συνάρτηση η οποία διπλασιάζει κάθε στοιχείο της λίστας σε μια λίστα αριθμών. Πρώτα, ας δημιουργήσουμε μια λίστα αριθμών:

```
my_list = [21, 2, 93]
```

Τώρα, ας φτιάξουμε τη συνάρτηση την οποία μπορούμε να ονομάσουμε `list_doubler`, αφού αυτό θέλουμε να κάνει. Σαν όρισμα, θα δέχεται τη λίστα της οποίας τα στοιχεία θα διπλασιάζει. Με όσα γνωρίζουμε μέχρι τώρα, θα γράφαμε:

```
def list_doubler(lst): # Συνάρτηση που διπλασιάζει τα
    στοιχεία μιας
    λίστας
    doubled = [] # Αρχικοποίηση μιας λίστας (κενή)
    for num in lst: # Για κάθε αριθμό στη λίστα όρισμα
        doubled.append(num*2) # πρόσθεσε στη νέα λίστα τον διπλάσιό
        του
    return doubled # Επέστρεψε τη λίστα
```

Καλώντας αυτή τη λίστα, θα είχαμε μια νέα με διπλασιασμένα στοιχεία.

```
my_doubled_list = list_doubler(lst)
```

δηλαδή, αν τρέχαμε το προγραμματάκι, θα είχαμε:

```
my_doubled_list = [42, 4, 186]
```

Η συνάρτηση είναι απλή και επιτυγχάνει αυτό που θέλουμε απλά, όμως καταλαμβάνει 5 γραμμές μαζί με τον ορισμό της συνάρτησης, έχει μια μεταβλητή με την οποία δεν κάνουμε τίποτα παρά να της προσθέσουμε τα στοιχεία και τέλος επιστρέφει «γεμάτη» από τα νέα στοιχεία.

Ο βρόχος for, επίσης δεν κάνει πολλά, απλά πολλαπλασιάζει έναν αριθμό με το 2. Έχουμε λοιπόν έναν κατάλληλο υποψήφιο για να εφαρμόσουμε το list comprehension.

Θα απλοποιήσουμε το εσωτερικό της συνάρτησης και όπως και πριν, όταν χρειαστεί, θα την καλέσουμε. Πρώτα απ' όλα, αφού οι list comprehensions δημιουργούν λίστες και οι λίστες μπορούν να ανατεθούν σε μεταβλητές, ας κρατήσουμε την doubled, αλλά ας βάλουμε τη list comprehension στα δεξιά της, δηλαδή;

```
doubled = [thing for thing in list_of_things]
```

Ωραία, τώρα χρειάζεται να συμπληρώσουμε τη δεξιά πλευρά. Όπως και σε όλους τους βρόχους for, με τους οποίους η δεξιά μεριά του comprehension list δείχνει ίδια, θα ονομάσουμε τα “things” στο βρόχο μας. Πρώτα, ας ονομάσουμε το καθένα κι επίσης θα χρησιμοποιήσουμε τη μεταβλητή στην οποία αναθέσαμε τη λίστα. Οπότε έχουμε:

```
doubled = [thing for num in lst]
```

Αυτό βέβαια δεν θα δουλέψει ακόμα, αφού το “thing” δεν είναι ακόμα “κάτι”. Στην αρχική μας συνάρτηση είχαμε num \* 2. έτσι, θα αντικαταστήσουμε μ' αυτό τώρα το “thing”:

```
doubled = [num * 2 for num in lst]
```

Οτιδήποτε είναι πριν το for, είναι αυτό που πραγματικά προστίθεται στη λίστα. Τέλος, “επιστρέφουμε” τη λίστα μας.

```
def list_doubler(lst):  
    doubled = [num * 2 for num in lst] #Διπλασίασε τον αριθμό,  
    για κάθε  
    αριθμό στη λίστα  
    return doubled
```

Καλούμε λοιπόν τη λίστα για να το δοκιμάσουμε:

```
my_doubled_list = list_doubler([12, 4, 202])
```

και βλέπουμε ότι η `my_doubled_list`, έχει τις αναμενόμενες τιμές `[24, 8, 404]`

Φαίνεται ότι δουλεύει, όμως, αφού μαζί με τη δημιουργία επιστρέφουμε κατευθείαν μια μεταβλητή, ας επιστρέψουμε τη `list comprehension` κατευθείαν:

```
def list_doubler(lst):  
    return [num * 2 for num in lst]
```

Οι `list comprehensions` χρησιμοποιούνται για να γλιτώσουμε χώρο. Επίσης, για να επεξεργαστούμε μια λίστα στα γρήγορα με μια επαναλαμβανόμενη εργασία. Τέλος, είναι χρήσιμες στον συναρτησιακό προγραμματισμό.

Όμως σε περίπτωση που όλα όσα θέλουμε να κάνουμε είναι να γίνουν σε μια λίστα, δεν θα μας χρησίμευαν και πολύ. Ευτυχώς μπορούν να χρησιμοποιηθούν με προϋποθέσεις.

## 32.2.1 List Comprehensions με if - else

Η list comprehension μας προσφέρει μια συντομότερη σύνταξη όταν θέλουμε να δημιουργήσουμε μια νέα λίστα με βάση τις τιμές μιας υπάρχουσας λίστας.

### Παράδειγμα 1:

Με βάση μια λίστα φρούτων, θέλουμε μια νέα λίστα, που να περιέχει μόνο τα φρούτα με το γράμμα "a" στην ονομασία του φρούτου.

Χωρίς list comprehension θα πρέπει να γράψουμε μια δήλωση for, με μια υπό όρους συνθήκη:

Κλασσική δήλωση:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
    if "a" in x:
        newlist.append(x)
print(newlist)
```

Με list comprehension μπορούμε να το κάνουμε όλο το παραπάνω σε μια και μόνο γραμμή.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

Η σύνταξη είναι:

**Νέα\_λίστα = [ενέργεια\_σε\_στοιχείο for στοιχείο in  
λίστα if συνθήκη == True]**

Η επιστρεφόμενη τιμή είναι μια λίστα αφήνοντας την αρχική λίστα ανέπαφη.

## Παράδειγμα 2

Επέστρεψε τα στοιχεία τα οποία δεν είναι 'apple'

```
newlist = [x for x in fruits if x != "apple"]
```

Η συνθήκη `if x != "apple"` είναι αληθής για όλα τα στοιχεία εκτός από το "apple", δηλαδή, η νέα λίστα περιέχει όλα τα φρούτα εκτός από τα μήλα.

## 32.2.2 Dictionary Comprehensions

Επεκτείνοντας την ιδέα, μπορούμε να δημιουργήσουμε και dictionary comprehensions, από άλλα λεξικά ή από το μηδέν.

Η σύνταξη είναι:

**Λεξικό\_εξόδου = {κλειδί:τιμή for κλειδί in iterable if (κλειδί, τιμή που ικανοποιεί τη συνθήκη)}**

## Παράδειγμα 1

Θέλουμε να δημιουργήσουμε ένα λεξικό το οποίο περιέχει μόνο

περιττούς αριθμούς σαν κλειδιά και τους κύβους τους σαν τιμές.

Ας δούμε πρώτα την κλασική προσέγγιση:

```
input_list = [1, 2, 3, 4, 5, 6, 7]
output_dict = {}
# Χρήση βρόχου for για δημιουργία του λεξικού εξόδου
for var in input_list:
    if var % 2 != 0:
        output_dict[var] = var**3
print(output_dict)
```

Με τη χρήση dictionary comprehension έχουμε:

```
input_list = [1,2,3,4,5,6,7]
dict_using_comp = {var:var ** 3 for var in input_list if var
% 2 != 0}
print(dict_using_comp)
```

Έξοδος:

{1: 1, 3: 27, 5: 125, 7: 343}

## Παράδειγμα 2

Μας δίνονται δύο λίστες, η μια από τις οποίες περιέχει τους νομούς και η άλλη τις πρωτεύουσες των νομών αντίστοιχα. Σχηματίστε ένα λεξικό το οποίο αποθηκεύει τους νομούς με τις αντίστοιχες πρωτεύουσές τους.

Κλασική προσέγγιση:

```
state = ['Αττική', 'Λέσβος', 'Πιερία']
capital = ['Αθήνα', 'Μυτιλήνη', 'Κατερίνη']
output_dict = {}
# Κατασκευή του λεξικού με for loop
for (key, value) in zip(state, capital):
    output_dict[key] = value
print("Το λεξικό είναι:", output_dict)
```

### Έξοδος:

Το λεξικό είναι: {'Αττική': 'Αθήνα', 'Λέσβος': 'Μυτιλήνη', 'Πιερία': 'Κατερίνη'}.

Πάμε τώρα να το δούμε με dictionary comprehension:

```
state = ['Αττική', 'Λέσβος', 'Πιερία']
capital = ['Αθήνα', 'Μυτιλήνη', 'Κατερίνη']
output_dict = {key:value for (key,value) in
zip(state,capital)}
print("Το λεξικό είναι:",output_dict)
```

### Παράδειγμα 3

Αντικαταστήστε κάθε τιμή του λεξικού με τη λέξη 'Odd' αν η τιμή είναι περιττός αριθμός, ή 'Even' αν είναι ζυγός.

```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
# Αναγνώριση περιττών και άρτιων στοιχείων
res = {k:('Even' if v % 2 == 0 else 'Odd') for (k, v) in
dict1.items()}
# Παρατηρήστε ότι χρειαζόμαστε και τη συνάρτηση items(),
ειδιάλλως
λαμβάνουμε σφάλμα στην έξοδο
print(res)
```

### Έξοδος:

{'a': 'Odd', 'b': 'Even', 'c': 'Odd', 'd': 'Even', 'e': 'Odd', 'f': 'Even'}

## 32.3.0 Lambda Functions

Στην Python, οι συναρτήσεις lambda, επίσης γνωστές ως ανώνυμες συναρτήσεις, είναι μικρές συναρτήσεις που δημιουργούνται inline και δεν απαιτούν μια τυπική δήλωση def. Χρησιμοποιούνται συχνά όταν

χρειαζόμαστε μια απλή συνάρτηση για ένα σύντομο χρονικό διάστημα, συνήθως ως όρισμα σε συναρτήσεις υψηλότερης προτεραιότητας ή σε καταστάσεις όπου η δημιουργία μιας ξεχωριστής συνάρτησης θα ήταν περιττή.

Ας δούμε έναν πίνακα με παραδείγματα:

<pre>def add(x, y):     return x + y</pre>	<pre>add = lambda x, y: x + y</pre>
<pre>def square(x):     return x ** 2</pre>	<pre>square = lambda x: x ** 2</pre>
<pre>def is_even(num):     return num % 2 == 0</pre>	<pre>is_even = lambda num: num % 2 == 0</pre>
<pre>def multiply(x, y, z):     return x * y * z</pre>	<pre>multiply = lambda x, y, z: x * y * z</pre>
<pre>def power(base, exponent):     return base ** exponent</pre>	<pre>power = lambda base, exponent: base ** exponent</pre>
<pre>def greet(name, greeting="Hello"):     return f'{greeting}, {name}!"</pre>	<pre>greet = lambda name, greeting="Hello": f'{greeting}, {name}!"</pre>

Ας θυμηθούμε και τα παρακάτω παραδείγματα.

## Παράδειγμα 1

Ανώνυμη συνάρτηση ως όρισμα σε μια άλλη συνάρτηση:

```
def process_numbers(numbers, func):  
    processed_numbers = []  
    for num in numbers:  
        processed_numbers.append(func(num))  
    return processed_numbers
```

και η func() θα μπορούσε να είναι μια συνάρτηση που επιστρέφει τα τετράγωνα αριθμών:

```
def func(x):
```



```
return x ** 2
```

Οπότε ένα πλήρες παράδειγμα θα ήταν:

```
def func(x):  
    return x ** 2  
# func = lambda x:x**2  
  
def process_numbers(numbers, func):  
    processed_numbers = []  
    for num in numbers:  
        processed_numbers.append(func(num))  
    return processed_numbers  
  
# Παράδειγμα χρήσης:  
numbers = [1, 2, 3, 4, 5]  
result = process_numbers(numbers, func)  
print(result)
```

Έξοδος: [1, 4, 9, 16, 25]

## Παράδειγμα 2

```
numbers = [1, 2, 3, 4, 5]  
result = process_numbers(numbers, lambda x: x ** 2)  
print(result)  
# Έξοδος: [1, 4, 9, 16, 25]
```

## 32.4.0 Αντικειμενοστραφής Προγραμματισμός

Ο αντικειμενοστραφής προγραμματισμός είναι μια παραδοχή ή μεθοδολογία προγραμματισμού που επιτρέπει την οργάνωση κώδικα σε αυτόνομες μονάδες που ονομάζονται "αντικείμενα". Ένα αντικείμενο είναι ένα πρότυπο ή μια "εκδοχή", ή ένα στιγμιότυπο μιας κλάσης και

περιέχει δεδομένα (μεταβλητές) και μεθόδους (συναρτήσεις) που δρουν πάνω σε αυτά τα δεδομένα.

### 32.4.1 Πλεονεκτήματα

- \* Ανακλαστικότητα (Reflection): Η δυνατότητα ενός αντικειμένου να "κατανοεί" τον εαυτό του και τις ιδιότητές του.
- \* Επαναχρησιμοποίηση κώδικα: Η δυνατότητα να δημιουργούμε κλάσεις και αντικείμενα που μπορούν να επαναχρησιμοποιηθούν σε διάφορα μέρη του προγράμματος.
- \* Κληρονομικότητα (Inheritance): Η δυνατότητα μιας κλάσης να κληρονομεί τις ιδιότητες και τις συμπεριφορές μιας άλλης κλάσης.
- \* Πολυμορφισμός (Polymorphism): Η δυνατότητα μιας μεθόδου να έχει διαφορετική συμπεριφορά, ανάλογα με τον τύπο του αντικειμένου που την καλεί.

### 32.4.2 Δημιουργία κλάσης – Μεταβλητές – Αρχικοποίηση

#### Ορισμός της κλάσης

Για να δημιουργήσουμε μια κλάση, χρησιμοποιούμε τη λέξη-κλειδί "class" ακολουθούμενη από το όνομα της κλάσης. Μέσα στην κλάση ορίζουμε τα χαρακτηριστικά και τις μεθόδους που θα έχει η κλάση.

#### Συναρτήσεις μέλη

Οι συναρτήσεις μέλη είναι μέθοδοι που ανήκουν σε μια κλάση. Αυτές

οι μέθοδοι μπορούν να εκτελέσουν ενέργειες σε αντικείμενα της κλάσης και να αλληλεπιδρούν με τα χαρακτηριστικά τους.

### **Μεταβλητές μέλη**

Οι μεταβλητές μέλη είναι μεταβλητές που ανήκουν σε μια κλάση. Αυτές οι μεταβλητές αποθηκεύουν δεδομένα για τα αντικείμενα της κλάσης και μπορούν να προσπελαστούν και να τροποποιηθούν από τις μεθόδους της κλάσης.

### **Αρχικοποίηση αντικειμένων**

Η αρχικοποίηση αντικειμένων γίνεται μέσω του κατασκευαστή (constructor) της κλάσης. Ο κατασκευαστής είναι μια ειδική μέθοδος που καλείται κατά τη δημιουργία ενός αντικειμένου και χρησιμοποιείται για την αρχικοποίηση των μεταβλητών μελών.

### **Πρόσβαση σε μέλη κλάσης**

Για να προσπελάσουμε τα μέλη μιας κλάσης, χρησιμοποιούμε τη σύνταξη "αντικείμενο.μέλος". Μπορούμε να αναφερθούμε στα χαρακτηριστικά ή να καλέσουμε τις μεθόδους του αντικειμένου.

## 32.5.0 Ασκήσεις

### Άσκηση 1

Χρησιμοποιήστε τη συνάρτηση `range()` για να δημιουργήσετε μια λίστα των αριθμών από 0 μέχρι 9. Ονομάστε τη λίστα `newlist` και χρησιμοποιήστε μόνο μια γραμμή κώδικα (list comprehension).

### Άσκηση 2

Χρησιμοποιήστε σαν δεδομένα τα παραπάνω, αλλά δεχτείτε μόνο αριθμούς μικρότερους από το 5.

### Άσκηση 3

Έστω η παρακάτω λίστα με ονόματα φρούτων:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
```

Μετατρέψτε τα στοιχεία της λίστας σε κεφαλαία γράμματα με μια και μόνο γραμμή κώδικα.

### Άσκηση 4

Αντικαταστήστε όλες τις τιμές των στοιχείων της λίστας με τη λέξη "hello":

### Άσκηση 5

Γράψτε μια γραμμή κώδικα για να πάρετε την παραπάνω λίστα, η οποία όμως να επιστρέφει "orange" αντί για "banana":

### Άσκηση 6

Φτιάξτε μια λίστα η οποία να τυπώνει όλα τα τετράγωνα των αριθμών από το 1 μέχρι και το 9.