7. ΕΡΓΑΣΙΑ ΜΕ ΣΥΝΑΡΤΗΣΕΙΣ – ΟΡΙΣΜΑΤΑ - ΛΙΣΤΕΣ

7.0 Λύσεις προηγούμενων ασκήσεων

1. Γράψτε μια συνάρτηση με το όνομα powerTo, η οποία όταν καλείται να ζητάει από τον χρήστη πρώτα έναν αριθμό (σαν βάση) και κατόπιν έναν άλλο για να τον υψώνει σε δύναμη. Τέλος, να τυπώνει το αποτέλεσμα.

Λύση:

```
def powerTo(x,y):
   x = int(input("Δώστε έναν αριθμό ο οποίος θα υψωθεί σε δύναμη: "))
   y = int(input("Τώρα δώστε τη δύναμη στην οποία θα τον υψώσουμε: "))
   print(x**y)

powerTo(2,3)
```

2. Γράψτε μια συνάρτηση countVowels η οποία να δέχεται μια λέξη ή πρόταση στα Ελληνικά και να μετράει πόσα φωνήεντα έχει. Στο τέλος να τυπώνει τον αριθμό φωνηέντων

Λύση:

```
def countVowels(string):
   vowels = 'αεηιυοάέίύόή'
   count = 0
   for char in string:
      if char.lower() in vowels:
         count += 1
   print("Η πρόταση σας έχει " + str(count) + " φωνήεντα")
```

countVowels("Σήμερα είναι μια ωραία μέρα")

3. Γράψτε μια συνάρτηση findLargest, η οποία θα δέχεται σαν παράμετρο μια λίστα αριθμών και όταν τρέχει, θα τυπώνει τον μεγαλύτερο απ' αυτούς

Λύση:

```
def findLargest(numbers):
    largest = numbers[0]
    for num in numbers:
        if num > largest:
            largest = num
    print(largest)

findLargest(numbers=[54,65,78,98,12,781,658,942])
```

4. Γράψτε μια συνάρτηση printEvens η οποία θα δέχεται σαν παράμετρο έναν φυσικό αριθμό και θα επιστρέφει όλους τους άρτιους από το 0 μέχρι τον αριθμό.

Λύση:

```
def printEvens(n):
    for i in range(n+1):
        if i % 2 == 0:
            print(i)

printEvens(92)
```

5. Γράψτε μια συνάρτηση filterEvens η οποία δέχεται μια λίστα αριθμών και τυπώνει στο τέλος μόνο τους ζυγούς

Λύση:

```
def filterEvens(numbers):

    evens = []

    for num in numbers:

        if num % 2 == 0:

            evens.append(num)
```

```
    print(evens)


filterEvens(numbers = [1,6,3,7,0,4,3,7,6,8,9])
```

7.1 Προαιρετικά ορίσματα (Optional arguments)

Κάποιες φορές είναι λογικό να κάνουμε ένα όρισμα προαιρετικό έτσι ώστε χρησιμοποιώντας τη συνάρτηση να μπορούμε να παρέχουμε κάποιο/α στοιχεία κατ' επιλογή. Μπορούμε να χρησιμοποιήσουμε προεπιλεγμένες τιμές για να κάνουμε ένα όρισμα προαιρετικό.

Για παράδειγμα, ας πούμε ότι θέλουμε να επεκτείνουμε το get_formatted_name() για χειρισμό και μεσαίων ονομάτων.

Μια πρώτη προσπάθεια να συμπεριλάβουμε μεσαία ονόματα θα μπορούσε να είναι:

```
def get_formatted_name(first_name, middle_name, last_name):
    """Επιστροφή πλήρους ονόματος και με το μεσαίο."""
    full_name = first_name + ' ' + middle_name + ' ' + last_name
    return full_name
musician = get_formatted_name('john', 'lee', 'hooker')
print(musician)
```

Αυτή η συνάρτηση λειτουργεί όταν δίνεται όνομα, μεσαίο όνομα και επίθετο. Παίρνει και τα τρία μέρη ενός ονόματος και στη συνέχεια δημιουργεί μια συμβολοσειρά από αυτά.
τους. Η συνάρτηση προσθέτει κενά όπου χρειάζεται και τυπώνει το πλήρες όνομα:

```
john lee hooker
```

Αλλά τα μεσαία ονόματα δεν χρειάζονται πάντα και η παραπάνω συνάρτηση δεν θα λειτουργούσε αν δίναμε μόνο δύο ορίσματα (όνομα και επώνυμο).

Για να κάνουμε το μεσαίο όνομα προαιρετικό, μπορούμε να δώσουμε στο όρισμα middle_name μια κενή προεπιλεγμένη τιμή και να αγνοήσουμε το όρισμα αν δοθούν μόνο τα άλλα δύο, ή να το τυπώσουμε αν ο χρήστης παρέχει μια τιμή.
Για να λειτουργήσει η get_formatted_name() χωρίς μεσαίο όνομα, ορίζουμε την προεπιλεγμένη τιμή της middle_name σαν κενή συμβολοσειρά και την μετακινούμε στο τέλος της λίστας παραμέτρων:

```
def get_formatted_name(first_name, last_name, middle_name=''):
    """ Επίστρεψε ένα πλήρες, καλοφορμαρισμένο όνομα."""
    if middle_name:
        full_name = first_name + ' ' + middle_name + ' ' + last_name
    else:
        full_name = first_name + ' ' + last_name
    return full_name.title() # Η title() μετατρέπει τα πρώτα γράμματα σε κεφαλαία

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

Έτσι θα πάρουμε:

Jimi Hendrix
John Lee Hooker

Σε αυτό το παράδειγμα, το όνομα δημιουργείται από τρία πιθανά μέρη. Επειδή υπάρχει πάντα ένα όνομα και ένα επίθετο, αυτές οι παράμετροι αναφέρονται πρώτες στον ορισμό της συνάρτησης.
Το μεσαίο όνομα είναι προαιρετικό, επομένως αναγράφεται τελευταίο στον ορισμό και η προεπιλεγμένη τιμή του είναι μια κενή συμβολοσειρά.
Στο σώμα της συνάρτησης, ελέγχουμε για να δούμε αν υπάρχει μεσαίο όνομα. Η Python ερμηνεύει τις μη κενές συμβολοσειρές ως True, οπότε αν το middle_name αξιολογείται σαν True τότε υπάρχει τιμή στο μεσαίο όνομα. Έτσι, συνδυάζονται όλα για να παρέχουν το πλήρες όνομα.
Αυτό στη συνέχεια αλλάζει σε τίτλο (κεφαλαία τα αρχικά) και επιστρέφει στη γραμμή κλήσης της συνάρτησης όπου αποθηκεύεται στη μεταβλητή musician και εκτυπώνεται.

Εάν δεν παρέχεται μεσαίο όνομα, η κενή συμβολοσειρά αποτυγχάνει στο if κι έτσι «τρέχει» το μπλοκ του else.
Το πλήρες όνομα διαμορφώνεται μόνο με όνομα και επίθετο, και το μορφοποιημένο όνομα επιστρέφεται στη γραμμή κλήσης, αποθηκεύεται και πάλι στη μεταβλητή musician και τυπώνεται.

Η κλήση αυτής της συνάρτησης με όνομα και επίθετο είναι απλή.
Αν όμως χρησιμοποιούμε μεσαίο όνομα, πρέπει να βεβαιωθούμε ότι είναι το θέσαμε στο τέλος, ώστε η Python να τρέξει σωστά, χωρίς μηνύματα σφάλματος.

αυτή η τροποποιημένη συνάρτηση λοιπόν τρέχει είτε δώσουμε μεσαίο όνομα είτε όχι.

Θα μπορούσαμε να την απλοποιήσουμε ακόμα περισσότερο, αφαιρώντας το if, else:

```
def get_formatted_name(first_name, last_name, middle_name=''):
    """ Επίστρεψε ένα πλήρες, καλοφορμαρισμένο όνομα."""
    full_name = first_name + ' ' + middle_name + ' ' + last_name
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

και θα παίρναμε το ίδιο αποτέλεσμα:

Jimi Hendrix
John Lee Hooker

Τα προαιρετικά ορίσματα επιτρέπουν στις συναρτήσεις να διαχειρίζονται πολλά use cases και ταυτόγχρονα οι δηλώσεις των συναρτήσεων παραμένουν απλές.

7.2 Επιστροφή ενός λεξικού (dictionary)

Μια συνάρτηση μπορεί να επιστρέψει οποιοδήποτε τύπο δεδομένων χρειαζόμαστε, συμπεριλαμβανομένων και πιο περίπλοκων δομών, όπως λίστες και λεξικά. Για παράδειγμα, παρακάτω η συνάρτηση παίρνει μέρη ενός ονόματος και επιστρέφει ένα λεξικό που αντιπροσωπεύει ένα άτομο:

```
def build_person(first_name, last_name):
    """Επίστρεψε το σύνολο πληροφοριών ενός ατόμου σε μορφή λεξικού."""
    person = {'first': first_name, 'last': last_name}
    return person
musician = build_person('jimi', 'hendrix')
print(musician)
```

Η συνάρτηση build_person() παίρνει ένα όνομα και ένα επίθετο και πακετάρει αυτές τις τιμές σε ένα λεξικό. Η τιμή του first_name αποθηκεύεται με το κλειδί 'first' και η τιμή του last_name αποθηκεύεται με το κλειδί 'last'. Ολόκληρο το λεξικό που αντιπροσωπεύει το άτομο (person) τυπώνεται με τα δύο πρωτότυπα στοιχεία κειμένου

τώρα να αποθηκεύονται σε ένα λεξικό:

```
{'first': 'jimi', 'last': 'hendrix'}
```

Αυτή η συνάρτηση λαμβάνει απλές πληροφορίες κειμένου και τις τοποθετεί σε μια πιο ουσιαστική δομή δεδομένων που μας επιτρέπει να εργαζόμαστε ίσως πιο αποδοτικά με τις πληροφορίες, πέρα από την απλή εκτύπωση.

Οι συμβολοσειρές «jimi» και «hendrix» ονομάζονται πλέον ως

ένα όνομα και επίθετο. Μπορούμε εύκολα να επεκτείνουμε αυτή τη λειτουργία για να δεχτούμε και προαιρετικές τιμές όπως μεσαίο όνομα, ηλικία, επάγγελμα ή οποιαδήποτε άλλη πληροφορία που θέλουμε να αποθηκεύσουμε για ένα άτομο.

Για παράδειγμα, με την εξής αλλαγή μπορούμε να αποθηκεύσουμε και την ηλικία ενός ατόμου:

```
def build_person(first_name, last_name, age=''):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
        return person


musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

Προσθέτουμε μια νέα προαιρετική παράμετρο, την ηλικία (age) στον ορισμό της συνάρτησης και αντιστοιχίζουμε στην παράμετρο μια κενή προεπιλεγμένη τιμή. Εάν η κλήση της συνάρτησης περιλαμβάνει τιμή για αυτήν την παράμετρο, η τιμή αποθηκεύεται στο λεξικό.

Αυτή η συνάρτηση αποθηκεύει πάντα το όνομα ενός ατόμου, αλλά μπορεί επίσης να τροποποιηθεί για να αποθηκεύσει οποιαδήποτε άλλη πληροφορία που θέλουμε για ένα άτομο.

7.3 Χρήση μιας συνάρτησης με το βρόχο while

Μπορούμε να χρησιμοποιήσουμε συναρτήσεις με όλες τις δομές της Python που έχουμε μάθει.
Για παράδειγμα, ας χρησιμοποιήσουμε τη συνάρτηση get_formatted_name() με ένα βρόχο while για να χαιρετάμε τους χρήστες πιο επίσημα με το όνομα και το επώνυμό τους:

```
def get_formatted_name(first_name, last_name):
    """ Επίστρεψε ένα πλήρες, καλοφορμαρισμένο όνομα."""
    full_name = first_name + ' ' + last_name
    return full_name.title()
# Αυτό το loop δεν τελειώνει ποτέ (infinite loop)!
while True:
    print("\nPlease tell me your name:")
    f_name = input("First name: ")
    l_name = input("Last name: ")
formatted_name = get_formatted_name(f_name, l_name)
print("\nHello, " + formatted_name + "!")
```

Με τον παραπάνω βρόχο while έχουμε ένα θέμα. Δεν έχουμε ορίσει τρόπο με τον οποίο να βγαίνουμε από αυτόν κι έτσι το πρόγραμμα, δεν σταματάει να ζητάει στοιχεία από τον χρήστη.
Πώς δίνουμε έναν τρόπο στον χρήστη να σταματήσει το πρόγραμμα να ζητάει στοιχεία όταν ζητάμε στοιχεία από αυτόν;
Θέλουμε ο χρήστης να μπορεί να σταματήσει το βρόχο όσο το δυνατόν πιο εύκολα, οπότε κάθε prompt θα πρέπει να προσφέρει έναν τρόπο διακοπής. Η break προσφέρει έναν απλό τρόπο εξόδου από τον βρόχο σε οποιαδήποτε από τις εντολές:

```
def get_formatted_name(first_name, last_name):
    """Επίστρεψε ένα πλήρες, καλοφορμαρισμένο όνομα"""
    full_name = first_name + ' ' + last_name
    return full_name.title()
while True:
    print("Παρακαλώ πείτε μου το όνομά σας:")
```

```
    print("(Πληκτρολογήστε το ';' και μετά 'Enter' για έξοδο)")
    f_name = input("Όνομα: ")

    if f_name == ';':
        break
    l_name = input("Επώνυμο: ")
    if l_name == ';':
        break

formatted_name = get_formatted_name(f_name, l_name)
print("Γεια σου, " + formatted_name + "!")
```

Έτσι, προσθέτουμε ένα μήνυμα που ενημερώνει τον χρήστη πώς να σταματήσει το βρόχο και μπορούμε να το κάνουμε σε οποιαδήποτε prompt.
Τώρα το πρόγραμμά μας τυπώνει:

```
Παρακαλώ πείτε μου το όνομά σας:
(Πληκτρολογήστε το ';' και μετά 'Enter' για έξοδο)
Όνομα: Κώστας
Επώνυμο: Αντωνίου
Γεια σου, Κώστας Αντωνίου!
Παρακαλώ πείτε μου το όνομά σας:
(Πληκτρολογήστε το ';' και μετά 'Enter' για έξοδο)
Όνομα: ;
```

7.4 Περνώντας μια λίστα σαν όρισμα

Συχνά θα μας φανεί χρήσιμο να περάσουμε μια λίστα σε μια συνάρτηση, είτε αυτή είναι μια λίστα με ονόματα ή αριθμούς είτε με πιο σύνθετα αντικείμενα, όπως είναι τα λεξικά.

Όταν περνάμε μια λίστα σε μια συνάρτηση, η συνάρτηση αποκτά άμεση πρόσβαση στα περιεχόμενα της λίστας. Αυτό, κάνει την εργασία με λίστες πιο αποτελεσματική.

Ας υποθέσουμε ότι έχουμε μια λίστα χρηστών και θέλουμε να εκτυπώσουμε έναν χαιρετισμό για τον καθένα ατομικά.

Το παρακάτω παράδειγμα στέλνει μια λίστα ονομάτων σε μια συνάρτηση που ονομάζεται greet_users(), που χαιρετά κάθε άτομο στη λίστα ξεχωριστά:

```
def greet_users(names):
    """ Εκτυπώστε έναν απλό χαιρετισμό σε κάθε χρήστη στη λίστα."""
    for name in names:
        msg = "Γεια σου, " + name.title() + "!"
        print(msg)


usernames = ['Μαρία', 'Ελένη', 'Νίκο']
greet_users(usernames)
```

Ορίζουμε την greet_users() ώστε να περιμένει μια λίστα ονομάτων, την οποία αποθηκεύει στην παράμετρο names. Η συνάρτηση κάνει βρόχο στη λίστα που λαμβάνει και τυπώνει έναν χαιρετισμό σε κάθε χρήστη. Κατόπιν, ορίζουμε μια λίστα χρηστών και μετά περνάμε τη λίστα usernames στην greet_users(), στην κλήση της συνάρτησης κι έχουμε:

Γεια σου, Μαρία!

Γεια σου, Ελένη!

Γεια σου, Νίκο!

Αυτό είναι το αποτέλεσμα που θα θέλαμε. Κάθε χρήστης βλέπει έναν εξατομικευμένο χαιρετισμό, και μπορούμε να καλέσουμε τη συνάρτηση οποιαδήποτε στιγμή θέλουμε να χαιρετίσουμε ένα συγκεκριμένο σύνολο χρηστών.

7.5 Τροποποίηση λίστας με συνάρτηση

Όταν περνάμε μια λίστα σε μια συνάρτηση, η συνάρτηση μπορεί να την τροποποιήσει. Οι αλλαγές που γίνονται στη λίστα μέσα στο σώμα της συνάρτησης είναι μόνιμες, επιτρέποντάς μας να εργαζόμαστε αποτελεσματικά ακόμα και όταν έχουμε να κάνουμε με μεγάλο όγκο δεδομένων.

Ας δώσουμε παρακάτω ένα παράδειγμα του πραγματικού κόσμου.

Ας υποθέσουμε ότι μια εταιρεία δημιουργεί τρισδιάστατα εκτυπωμένα μοντέλα σχεδίων που οι χρήστες υποβάλλουν.

Τα σχέδια που πρέπει να εκτυπωθούν αποθηκεύονται σε μια λίστα και μετά την εκτύπωση, μεταφέρονται σε ξεχωριστή λίστα.

Ο παρακάτω κώδικας το κάνει αυτό χωρίς τη χρήση συναρτήσεων:

```
# Ξεκινάμε με μερικά σχέδια που πρέπει να εκτυπωθούν.
unprinted_designs = ['θήκη iPhone', 'ρομποτικό μενταγιόν', 'δωδεκάεδρο']
completed_models = []
# Προσομοιώνουμε την εκτύπωση κάθε σχεδίου, μέχρι να μην μείνει κανένα.
# Μετακινούμε κάθε σχέδιο στα completed_models μετά την εκτύπωση.
while unprinted_designs:
    current_design = unprinted_designs.pop()
    # Προσομοιώνουμε την εκτύπωση ενός τρισδιάστατου σχεδίου.
    print("Printing model: " + current_design)
    completed_models.append(current_design)
# Εμφάνισε όλα τα ολοκληρωμένα μοντέλα.
print("Τα παρακάτω μοντέλα έχουν τυπωθεί:")
```

for completed_model in completed_models:

print(completed_model)

Αυτό το πρόγραμμα ξεκινάει με μια λίστα σχεδίων που πρέπει να εκτυπωθούν και με μια κενή λίστα που ονομάζεται completed_models στην οποία θα μετακινηθεί κάθε σχέδιο που έχει τυπωθεί. Όσο τα σχέδια παραμένουν στην unprinted_designs, ο βρόχος while προσομοιώνει την εκτύπωση κάθε σχεδίου αφαιρώντας ένα σχέδιο από το τέλος της λίστας, αποθηκεύοντάς την στο current_design και εμφανίζοντας ένα μήνυμα ότι το τρέχον σχέδιο τυπώνεται.

Στη συνέχεια προσθέτει το σχέδιο στη λίστα των completed_models. Όταν ολοκληρωθεί η εκτέλεση του βρόχου, εμφανίζεται η λίστα με τα έτοιμα μοντέλα (completed_models):

Printing model: δωδεκάεδρο

Printing model: ρομποτικό μενταγιόν

Printing model: θήκη iPhone

Τα παρακάτω μοντέλα έχουν τυπωθεί:

δωδεκάεδρο

ρομποτικό μενταγιόν

θήκη iPhone

Μπορούμε να αναδιοργανώσουμε αυτόν τον κώδικα γράφοντας δύο συναρτήσεις, καθεμία από τις οποίες κάνει μια συγκεκριμένη δουλειά. Το μεγαλύτερο μέρος του κώδικα δεν θα αλλάξει. απλά το φτιάχνουμε

πιο αποτελεσματικό. Η πρώτη συνάρτηση θα χειριστεί την εκτύπωση των σχεδίων και η δεύτερη θα συνοψίσει τις εκτυπώσεις που έχουν γίνει:

```
def print_models(unprinted_designs, completed_models):
    # Προσομοιώνουμε την εκτύπωση κάθε σχεδίου, μέχρι να μην μείνει κανένα.
    # Μετακινούμε κάθε σχέδιο στα completed_models μετά την εκτύπωση.
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        # Προσομοιώνουμε την εκτύπωση ενός τρισδιάστατου σχεδίου.
        print("Τυπώνεται το μοντέλο: " + current_design)
        completed_models.append(current_design)
def show_completed_models(completed_models):
    # Εμφάνισε όλα τα ολοκληρωμένα μοντέλα.
    print("Τα παρακάτω μοντέλα έχουν τυπωθει:")
    for completed_model in completed_models:
        print(completed_model)
unprinted_designs = ['θήκη iphone', 'ρομποτικό μενταγιόν', 'δωδεκάεδρο']
completed_models = []
print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Εδώ, ορίζουμε τη συνάρτηση print_models() με δύο παραμέτρους: μια λίστα των σχεδίων που πρέπει να εκτυπωθούν και μια λίστα με ολοκληρωμένα μοντέλα. Δεδομένων αυτών των δύο λιστών, η συνάρτηση προσομοιώνει την εκτύπωση κάθε σχεδίου αδειάζοντας τη λίστα ατύπωτων σχεδίων και γεμίζοντας τη λίστα ολοκληρωμένων μοντέλων. Κατόπιν, ορίζουμε τη συνάρτηση show_completed_models() με μία παράμετρο: τη λίστα των ολοκληρωμένων μοντέλων.

Δεδομένης αυτής της λίστας, η show_completed_models() εμφανίζει το όνομα κάθε μοντέλου που τυπώθηκε.

Αυτό το πρόγραμμα έχει την ίδια έξοδο με την έκδοση χωρίς συναρτήσεις, αλλά ο κώδικας είναι πολύ πιο οργανωμένος. Ο κώδικας που κάνει το μεγαλύτερο μέρος της δουλειάς έχει μετακινηθεί σε δύο ξεχωριστές συναρτήσεις, γεγονός που καθιστά το κύριο μέρος του

προγράμματος πιο κατανοητό.

Ας δούμε το σώμα του προγράμματος για να δούμε πώς

είναι πολύ πιο εύκολο να καταλάβουμε τι κάνει αυτό το πρόγραμμα:

unprinted_designs = ['θήκη iphone', 'ρομποτικό μενταγιόν', 'δωδεκάεδρο']

completed_models = []

print_models(unprinted_designs, completed_models)

show_completed_models(completed_models)

Δημιουργήσαμε μια λίστα με μη εκτυπωμένα σχέδια και μια κενή λίστα που θα περιέχει τα ολοκληρωμένα μοντέλα. Στη συνέχεια, επειδή έχουμε ήδη ορίσει τις δύο συναρτήσεις μας, το μόνο που έχουμε να κάνουμε είναι να τις καλέσουμε και να τις δώσουμε τα σωστά ορίσματα.

Καλούμε την print_models() και της περνάμε τις δύο λίστες που χρειάζεται. Όπως αναμενόταν, η print_models() προσομοιώνει την εκτύπωση των σχεδίων.

Στη συνέχεια καλούμε την show_completed_models() και της περνάμε μια λίστα με τα ολοκληρωμένα μοντέλα ώστε να μπορεί να αναφέρει τα μοντέλα που έχουν τυπωθεί. Τα περιγραφικά ονόματα των συναρτήσεων επιτρέπουν την εύκολη κατανόηση του τι κάνουν οι συναρτήσεις, ίσως ακόμα και χωρίς σχόλια.

Αυτό το πρόγραμμα είναι πιο εύκολο να επεκταθεί και να συντηρηθεί, από την έκδοση χωρίς συναρτήσεις. Αν χρειαστεί να εκτυπώσουμε περισσότερα σχέδια αργότερα, μπορούμε απλά να καλέσουμε την

print_models() ξανά. Εάν καταλάβουμε ότι ο κώδικας εκτύπωσης πρέπει να τροποποιηθεί, μπορούμε να αλλάξουμε τον κώδικα μία φορά και οι αλλαγές μας θα πραγματοποιηθούν οπουδήποτε καλείται η συνάρτηση.

Αυτή η τεχνική είναι πιο αποτελεσματική από την ανάγκη ενημέρωσης του κώδικα χωριστά σε πολλά σημεία του προγράμματος.

Αυτό το παράδειγμα δείχνει επίσης την ιδέα ότι κάθε συνάρτηση πρέπει να έχει μια συγκεκριμένη δουλειά. Η πρώτη συνάρτηση εκτυπώνει κάθε σχέδιο και η δεύτερη εμφανίζει τα ολοκληρωμένα μοντέλα.

Αυτό είναι πιο χρήσιμο, κατανοητό και ευκολοσυντήρητο από τη χρήση μιας συνάρτησης που κάνει και τις δύο δουλειές.

Έτσι αν γράφουμε μια συνάρτηση και παρατηρήσουμε ότι κάνει πάρα πολλές διαφορετικές εργασίες, προσπαθούμε να χωρίσουμε τον κώδικα σε δύο ή και περισσότερες συναρτήσεις.

Να θυμηθούμε ότι μπορούμε να καλέσουμε μια συνάρτηση μέσα από μια άλλη συνάρτηση, κι αυτό μπορεί να είναι χρήσιμο όταν χωρίζουμε μια σύνθετη εργασία σε μια σειρά βημάτων.

7.6 Αποτροπή τροποποίησης λίστας από συνάρτηση

Μερικές φορές θα θέλαμε να αποτρέψουμε την τροποποίηση μιας λίστας. Για παράδειγμα, ας πούμε ότι ξεκινάμε με μια λίστα μη εκτυπωμένων σχεδίων και γράφουμε μια συνάρτηση για να τα μετακινήσουμε σε μια λίστα ολοκληρωμένων μοντέλων, όπως στο προηγούμενο παράδειγμα.

Μπορεί να αποφασίσουμε ότι παρόλο που έχουμε τυπώσει όλα τα σχέδια, θέλουμε να διατηρήσουμε την αρχική λίστα των μη εκτυπωμένων σχεδίων για το αρχείο μας.

Αλλά επειδή μετακινήσαμε όλα τα ονόματα σχεδίων από το ***unprinted_designs***, η λίστα είναι τώρα κενή και η κενή λίστα είναι η μόνη έκδοση που έχουμε. Το πρωτότυπο έχει χαθεί.

Σε αυτήν την περίπτωση, μπορούμε να αντιμετωπίσουμε αυτό το θέμα περνώντας στη συνάρτηση ένα αντίγραφο της λίστας, όχι την αρχική. Οποιεσδήποτε αλλαγές κάνει η συνάρτηση στη λίστα θα επηρεάζουν μόνο το αντίγραφό της, αφήνοντας ανέπαφη την αρχική λίστα.

Μπορείτε να στείλουμε ένα αντίγραφο μιας λίστας σε μια συνάρτηση έτσι:

όνομα_συνάρτησης(όνομα_λίστας[:])

Το slice notation [:] δημιουργεί ένα αντίγραφο της λίστας για πέρασμα στη συνάρτηση.

Αν δεν θέλαμε να αδειάσουμε τη λίστα των μη εκτυπωμένων σχεδίων στο print_models.py, θα μπορούσαμε να καλέσουμε την print_models() ως εξής:

print_models(unprinted_designs[:], completed_models)

Η συνάρτηση ***print_models()*** μπορεί να κάνει τη δουλειά της επειδή εξακολουθεί να λαμβάνει τα ονόματα όλων των ατύπωτων σχεδίων. Αλλά αυτή τη φορά χρησιμοποιεί ένα αντίγραφο της πρωτότυπης

λίστας μη εκτυπωμένων σχεδίων, κι όχι την πραγματική λίστα unprinted_designs. Η λίστα ***completed_models*** θα γεμίσει με τα ονόματα των εκτυπωμένων μοντέλων όπως έγινε πριν, αλλά η αρχική λίστα των μη εκτυπωμένων σχεδίων δεν θα επηρεαστεί από τη

συνάρτηση.

Παρά το ότι μπορούμε να διατηρήσουμε τα περιεχόμενα μιας λίστας περνώντας ένα αντίγραφο στις συναρτήσεις μας, θα πρέπει να περνάμε την αρχική λίστα στις συναρτήσεις εκτός εάν έχουμε συγκεκριμένο λόγο να περνάμε ένα αντίγραφο.

Είναι πιο αποτελεσματικό για μια συνάρτηση δουλεύει με μια αρχική λίστα γιατί έτσι αποφεύγουμε τη χρήση του χρόνου και της μνήμης που απαιτούνται, ειδικά όταν εργαζόμαστε με μεγάλες λίστες.

7.8 Ασκήσεις

1. Κάντε μια λίστα με ονόματα καλλιτεχνών (καλύτερα ξένων). Περάστε τη λίστα σε μια συνάρτηση που ονομάζεται show_artists(), η οποία τυπώνει το όνομα κάθε καλλιτέχνη στη λίστα.

2. Ξεκινήστε με ένα αντίγραφο του προγράμματός σας από την προηγούμενη άσκηση. Γράψτε μια συνάρτηση που ονομάζεται make_great() που τροποποιεί τη λίστα των καλλιτεχνών προσθέτοντας τη φράση «The Great» στο όνομα κάθε καλλιτέχνη. Καλέστε την show_artists() και δείτε ότι η λίστα έχει όντως τροποποιηθεί.

3. Ξεκινήστε με την εργασία σας από την προηγούμενη άσκηση. Καλέστε τη συνάρτηση make_great() με ένα αντίγραφο της λίστας των ονομάτων των καλλιτεχνών. Επειδή η αρχική λίστα θα παραμείνει αμετάβλητη, επιστρέψτε τη νέα λίστα και αποθηκεύστε την σε ξεχωριστή λίστα. Καλέστε τη show_artists() με κάθε λίστα για να δείξετε ότι έχετε μία λίστα με τα αρχικά ονόματα και μία λίστα με το «The Great» να προστίθεται στο όνομα κάθε καλλιτέχνη.

4. Γράψτε μια συνάρτηση που ονομάζεται describe_city() που δέχεται το όνομα μιας πόλης της Ελλάδας και το νομό της. Η συνάρτηση θα πρέπει να εκτυπώνει μια απλή πρόταση, όπως “Η Κατερίνη βρίσκεται στο νομό Πιερίας”. Δώστε στην παράμετρο για το νομό μια προεπιλεγμένη τιμή. Καλέστε τη συνάρτησή σας για τρεις διαφορετικές πόλεις, τουλάχιστον μία από τις οποίες δεν βρίσκεται στον προεπιλεγμένο νομό.

5. Ζητήστε από τον χρήστη έναν ακέραιο n και γράψτε ένα πρόγραμμα το οποίο δημιουργεί ένα λεξικό που περιέχει το “i” και το τετράγωνό του “i*i”, δηλ, το λεξικό να έχει τα ζεύγη (i, i * i) από το 1 μέχρι και τον ακέραιο n ο οποίος ζητείται (και τα δύο περιλαμβάνονται).

Κατόπιν, το πρόγραμμα θα πρέπει να εκτυπώνει το λεξικό.

6. Γράψτε ένα πρόγραμμα που ρωτά τον χρήστη πόσα άτομα είναι στην παρέα του για να κλείσουν εισιτήρια για το θέατρο . Εάν η απάντηση είναι μεγαλύτερη από οκτώ, εκτυπώστε ένα μήνυμα λέγοντας ότι θα

πρέπει να καθίσουν ξεχωριστά. Διαφορετικά, αναφέρετε ότι μπορούν να κλείσουν θέσεις.