10. PYTHON – ΕΝΑΛΛΑΚΤΙΚΗ ΣΥΝΤΑΞΗ

10.1.0 Shorthand if… else

Αν έχουμε μόνο μια δήλωση για εκτέλεση, μπορούμε να τη βάλουμε στην ίδια γραμμή με την if… else.

Παράδειγμα:

```
if x > y: print("x is greater than y")
```

10.2.0 Τριαδικός τελεστής

Δήλωση if… else με τρεις συνθήκες σε μια γραμμή.

Παράδειγμα:

```
x = 330
y = 330
print("X") if x > y else print("=") if x == y else print("Y")
```

10.3.0 List Comprehension

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 μοιάζουνε 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 # Επίστρεψε 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.

10.3.1 Δημιουργία 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.

10.3.2 List comprehensions με δηλώσεις υπό συνθήκη - 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)
```

List comprehension

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.

10.3.3 Iterable - Περιέκτης αντικειμένων, “μέσα” στον οποίο γίνεται η επανάληψη του βρόχου

Το Iterable (διατρέξιμο)

μπορεί να είναι οποιοδήποτε αντικείμενο μέσα στο οποίο μπορεί να “τρέξει” ένας βρόχος, όπως μια λίστα, ένα tuple, ένα set κ.λ.π.

Παράδειγμα 3

Μπορούμε να χρησιμοποιήσουμε τη συνάρτηση range() για να δημιουργήσουμε ένα iterable:

```
newlist = [x for x in range(10)]
```

Δημιουργήσαμε μια λίστα από το 0 μέχρι και το 9 και της έχουμε δώσει και όνομα σε μια και μόνο γραμμή κώδικα.

Αν πάρουμε μια εκτύπωση της newlist θα έχουμε:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Παράδειγμα 4

Δέξου μόνο αριθμούς μικρότερους από το 5:

```
newlist = [x for x in range(10) if x < 5]
```

Ζητώντας την εκτύπωση της παραπάνω λίστας έχουμε:

```
[0, 1, 2, 3, 4]
```

Παράδειγμα 5

Μετέτρεψε τα στοιχεία της νέας λίστας σε κεφαλαία (από τη λίστα των φρούτων) :

newlist = [x.upper() for x in fruits]

Δηλαδή:

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

newlist = [x.upper() for x in fruits]

print(newlist)
```

Έξοδος:

```
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

Παράδειγμα 6

Αντικατέστησε όλες τις τιμές των στοιχείων της λίστας με τη λέξη “hello”:

newlist = ['hello' for x in fruits]

Δηλαδή:

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

newlist = ['hello' for x in fruits]

print(newlist)
```

Έξοδος:

```
['hello', 'hello', 'hello', 'hello', 'hello']
```

Η κάθε δήλωση, μπορεί επίσης να περιέχει συνθήκες, όχι σαν ένα φίλτρο, αλλά σαν ένα τρόπο να διαχειριστούμε το αποτέλεσμα της εξόδου:

Παράδειγμα 7

Επίστρεψε "orange" αντί για "banana":

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

Δηλαδή:

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

newlist = [x if x != "banana" else "orange" for x in fruits]

print(newlist)
```

Έξοδος:

```
['apple', 'orange', 'cherry', 'kiwi', 'mango']
```

Η παραπάνω έκφραση λέει:

Επίστρεψε κάθε φρούτο αν δεν είναι μπανάνα, όμως αν είναι μπανάνα, επίστρεψε πορτοκάλι

Παράδειγμα 9

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

Λύση:

```
squares_lst = [x**2 for x in range(10)]

print(squares_lst)
```

Έξοδος:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Παράδειγμα 10:

Ας φτιάξουμε μια νέα συνάρτηση η οποία μας επιστρέφει μόνο τις μεγάλες λέξεις από μια λίστα με λέξεις. Μπορούμε να πούμε πως οποιαδήποτε λέξη μεγαλύτερη από 7 χαρακτήρες είναι μια μεγάλη λέξη. Ας τη γράψουμε πρώτα με τον κλασικό τρόπο:

```
def long_words(lst):
    words = []
    for word in lst:
        if len(word) > 5:
            words.append(word)
    return words
```

Τώρα, αναθέτουμε σε μια μεταβλητή να κρατήσει τις λέξεις μας και κάνουμε loop σε όλες τις λέξεις στη λίστα για να ελέγξουμε το μήκος τους. Αν είναι μεγαλύτερο από 7 όπως είπαμε, προσθέτουμε τη λέξη στη λίστα και τέλος, επιστρέφουμε τη λίστα. Έτσι η λίστα:

```
long_words=['blog', 'Treehouse', 'Python_Rules', 'hi'])
```

επιστρέφει:

```
['Treehouse', 'Python_Rules']
```

δηλαδή το αναμενόμενο.

Ας ξαναγράψουμε τώρα τη συνάρτηση χρησιμοποιώντας τη σύνταξη list comprehension:

```
def long_words(lst):

    return [word for word in lst]
```

Μπορούμε να χρησιμοποιήσουμε τον παραπάνω κώδικα σαν ένα ενδιάμεσο βήμα, αφού μας επιστρέφει όλη τη λίστα. Τώρα μπορούμε να βάλουμε και τη δήλωση υπό συνθήκη στο τέλος του βρόχου for):

```
def long_words(lst):

    return [word for word in lst if len(word) > 5]
```

Δοκιμάζουμε τον κώδικά μας χρησιμοποιώντας την παρακάτω λίστα:

```
long_words(['list', 'comprehension', 'Treehouse', 'Ken'])
```

η οποία μας επιστρέφει:

```
['comprehension', 'Treehouse'].
```

10.4.0 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 ‘Odd’ αν είναι ζυγός.

```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}

# Αναγνώριση περιττών και ζυγών στοιχείων

res = {k:('Even' if v % 2 == 0 else 'Odd') for (k, v) in dict1.items()}

# Παρατηρήστε ότι χρειαζόμαστε και τη συνάρτηση items(), ειδάλλως λαμβάνουμε σφάλμα στην έξοδο

print(res)
```

Έξοδος:

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

Παράδειγμα 4

Δημιουργία λεξικού με κλειδιά που είναι γράμματα του αλφαβήτου και αύξοντες αριθμούς σαν τιμές. Για να πάρουμε τα absde χρειαζόμαστε την έκφραση:

```
string.ascii_lowercase[:5]
```

Έτσι έχουμε:

```
import string

{i:j for (i, j) in zip(string.ascii_lowercase[:5], range(5))}
```

Έξοδος:

{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}

Παράδειγμα 5

Η dic.items() επιστρέφει όλη τη δομή ενός λεξικού με κλειδιά και τιμές. Στο παρακάτω παράδειγμα πολλαπλασιάζουμε τις τιμές ενός υπαρκτού λεξικού με το 2 και παίρνουμε ένα νέο λεξικό new_dic (όπως κάναμε και με τις λίστες παραπάνω)

```
dic = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

new_dic = {i:j*2 for i,j in dic.items()}

print(new_dic)
```

Έξοδος:

{'a': 2, 'b': 4, 'c': 6, 'd': 8}

Παράδειγμα 6

Μετατροπή κιλών σε pounds σε ζευγάρια τιμών σε λεξικό:

```
old_weights = {
    'book': 0.5,
    'milk': 2.0,
    'tv': 7.0
}

new_weights = {key: value*2.2 for (key, value) in old_weights.items()}

print(new_weights)
```

Έξοδος:

{'book': 1.1, 'milk': 4.4, 'tv': 15.400000000000002}

10.5.0 Ασκήσεις

Άσκηση 1 (generate_dict_n.py)

Ζητήστε από τον χρήστη έναν ακέραιο αριθμό n. Γράψτε ένα πρόγραμμα για τη δημιουργία ενός λεξικού που περιέχει σαν κλειδί έναν αριθμό 'I' και σαν τιμή τον i * i (δηλ. το τετράγωνό του i x i) τέτοιο ώστε να είναι ένας ακέραιος αριθμός μεταξύ του 1 και του n (και τα δύο περιλαμβάνονται) που έχει δοθεί από τον χρήστη. και μετά το πρόγραμμα θα πρέπει να εκτυπώσει το λεξικό. Αν για παράδειγμα δώσουμε τον αριθμό 7, θα πρέπει να πάρουμε σαν έξοδο:

{1: 1}

{1: 1, 2: 4}

{1: 1, 2: 4, 3: 9}

{1: 1, 2: 4, 3: 9, 4: 16}

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}

Κατόπιν, μετατρέψτε τον κώδικα σε dictionary comprehension.

Άσκηση 2 (print_dict.py)

Γράψτε μια συνάρτηση η οποία μπορεί να τυπώνει ένα λεξικό όπου τα κλειδιά είναι οι αριθμοί από το 1 μέχρι και το 20 και οι τιμές είναι τα τετράγωνά τους. Χρησιμοποιήστε την dictionary comprehension. Αν όλα πάνε σωστά, η εκτύπωσή σας θα είναι:

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225, 16: 256, 17: 289, 18: 324, 19: 361, 20: 400}

Άσκηση 3 (count_freq.py)

Σας δίνεται μια συμβολοσειρά. Πρέπει να μετρήσετε τη συχνότητα των γραμμάτων της συμβολοσειράς και να τυπώσετε τον αριθμό των γραμμάτων με φθίνουσα σειρά συχνότητας.

Αν δοθεί για παράδειγμα η συμβολοσειρά: ***aabbbccde***, η έξοδος του προγράμματος πρέπει να είναι: ***b 3 a 2 c 2 d 1 e 1***

(Χρησιμοποιείστε Αγγλικές φράσεις. Θα χρειαστεί να χρησιμοποιήσετε λεξικά, καθώς και τη συνάρτηση sorted() για να ταξινομήσετε το αντίστοιχο λεξικό με φθίνουσα σειρά συχνότητας, όπως ζητείται. Μην προσπαθήσετε να χρησιμοποιήσετε dictionary comprehension εδώ, καθότι θα χρειαστεί επίσης να χρησιμοποιήσετε lamda functions που δεν έχουμε διδαχθεί).