30.ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΧΡΗΣΤΗ – IF, ELIF, ELSE – PRINT - ΣΥΝΑΡΤΗΣΕΙΣ

30.0.1 Λύσεις των προηγούμενων ασκήσεων

1. Γράψτε το παρακάτω ποίημα ώστε σαν έξοδο να έχετε την παρακάτω εκτύπωση:

Λύση

```
print("Όταν ήμουν μικρό παιδί, όπως είναι\n τ’ αρνάκια
πού παίζουνε\n στό λειβάδι τήν άνοιξη,\n\t είχα φίλους
τίς πασχαλίτσες\n\t\t  είχα φίλους τίς
λιμπελούλες\n\t\t\t είχα φίλους τά λουλουδάκια.\n\t\t\t
Νικηφόρος Βρεττάκος")
```

2. Γράψτε ένα πρόγραμμα που να σας δείχνει ποια έκδοση της Python έχετε (όχι στο command line).

Λύση

```
import sys
print("Python version")
print (sys.version)
#print("Version info.")
#print (sys.version_info)
```

3. Γράψτε ένα πρόγραμμα που να σας δείχνει την ημερομηνία και την ώρα αυτή τη στιγμή.

Λύση

```
import datetime
now = datetime.datetime.now()
print("Η ημερομηνία και η ώρα είναι: ")
# print(now.strftime("%Y-%m-%d %H:%M:%S"))
print(now.strftime("%d-%m-%Y %S:%M:%H"))
```

4. Γράψτε ένα πρόγραμμα που θα δέχεται έναν αριθμό δευτερολέπτων και θα υπολογίζει πόσα λεπτά και δευτερόλεπτα αντιστοιχούν.

Λύση

```
# Υπολογισμός του Χρόνου σε Λεπτά και Δευτερόλεπτα
total_seconds = int(input("Εισάγετε τον συνολικό αριθμό
δευτερολέπτων: "))
minutes = total_seconds // 60
seconds = total_seconds % 60
print(f"{total_seconds} δευτερόλεπτα είναι ίσα με
{minutes} λεπτά και {seconds}
```

5. Γράψτε ένα πρόγραμμα που θα δέχεται έναν ακέραιο από τον χρήστη και θα ελέγχει αν είναι ζυγός ή περιττός χρησιμοποιώντας τον τελεστή %.

Λύση

```
# Αναζήτηση Ζυγών Αριθμών
number = int(input("Εισάγετε έναν αριθμό: "))
if number % 2 == 0:
    print(f"Ο αριθμός {number} είναι ζυγός.")
else:
    print(f"Ο αριθμός {number} είναι περιττός.")
```

6. Γράψτε ένα πρόγραμμα που θα δέχεται δύο αριθμούς από το χρήστη και θα υπολογίζει το υπόλοιπο της διαίρεσης του πρώτου αριθμού από τον δεύτερο χρησιμοποιώντας τον τελεστή %.

```
# Υπολογισμός Υπολοίπου Διαίρεσης
```

```
numerator = int(input("Εισάγετε τον αριθμητή (μεγαλύτερο αριθμό): "))
denominator = int(input("Εισάγετε τον παρονομαστή (μικρότερο αριθμό): "))
remainder = numerator % denominator
print(f"Το υπόλοιπο της διαίρεσης {numerator} // {denominator} είναι {remainder}")
```

7. Γράψτε ένα προγραμματάκι το οποίο τυπώνει το ημερολόγιο ενός μήνα κάποιου έτους. Ζητάει και τα δύο στοιχεία από τον χρήστη (πρώτα το έτος). Εισάγετε τη βιβλιοθήκη calendar.

```
import calendar
y = int(input("Εισάγετε το έτος : "))
m = int(input("Εισάγετε τον μήνα : "))
print(calendar.month(y, m))
```

8. Γράψτε ένα πρόγραμμα το οποίο να προσθέτει τα στοιχεία μιας λίστας το ένα μετά το άλλο, ακόμα κι αν είναι αριθμητικά. Δώστε 2 παραδείγματα, ένα με αριθμούς κι ένα με συμβολοσειρές. Η εκτύπωση πρέπει να είναι:

```
= RESTART: C:/Users
15122
Η Python σκίζει
```

```
def concatenate_list_data(list):
    result= ''
    for element in list:
        result += str(element)
    return result

print(concatenate_list_data([1, 5, 12, 2]))

print(concatenate_list_data(["Η ", "Python ", "σκίζει"]))
```

30.1.0 Διαχείριση λιστών (Ανεύρεση, προσθήκη, διαγραφή, προσπέλαση στοιχείων)

Έστω η λίστα userAge = [21, 3, 23, 24, 25].

Παραδείγματα συμβολισμών slicing:

Το userAge[:4] μας δίνει τιμές από το 0 έως το index 4-1 ενώ το userAge[1:] μας δίνει τιμές από το index 1 έως το index 5-1 (αφού

το μέγεθος του userAge είναι 5, δηλαδή το userAge έχει 5 στοιχεία.

Για να τροποποιήσουμε στοιχεία σε μια λίστα, γράφουμε listName[index του στοιχείου που θα τροποποιηθεί] = νέα τιμή.Για

παράδειγμα, εάν θέλουμε να τροποποιήσουμε το δεύτερο στοιχείο,

γράφουμε userAge[1] = 5.

Η λίστα μας γίνεται userAge = [21, 5, 23, 24, 25].

Για να προσθέσουμε στοιχεία, χρησιμοποιούμε τη συνάρτηση append(). Για παράδειγμα, αν γράψουμε:

userAge.append(99), προσθέτουμε την τιμή 99 στο τέλος της λίστας.

Η λίστα μας γίνεται:

userAge = [21, 5, 23, 24, 25, 99].

Για να αφαιρέσουμε στοιχεία, γράφουμε:

del listName[index του στοιχείου που θα είναι προς διαγραφή].Για

παράδειγμα:

del userAge[2]

η λίστα τώρα γίνεται userAge = [21, 5, 24, 25, 99] (το τρίτο στοιχείο διαγράφεται)

Εκτελέσετε το παρακάτω πρόγραμμα στον IDLE:

```
#δήλωση της λίστας, τα στοιχεία της λίστας μπορεί να
έχουν διαφορετικούς τύπους δεδομένων
myList = [1, 2, 3, 4, 5, "Γεια"] #εκτύπωση ολόκληρης της
λίστας. print (myList)
#Θα λάβουμε [1, 2, 3, 4, 5, "Γεια"]
#print το τρίτο στοιχείο (υπενθύμιση: Το ευρετήριο ξεκινά
από το μηδέν).
print(myList[2])
#Θα πάρουμε 3
#εκτύπωση του τελευταίου στοιχείου.
print(myList[-1])
#Θα λάβουμε "Γεια"
#assign myList (από το index 1 έως το 4) στο myList2 και
εκτύπωση
myList2
myList2 = myList[1:5] εκτύπωση (myList2) #Θα λάβουμε [2,
3, 4, 5]
#modify το δεύτερο στοιχείο στο myList και εκτύπωση της
ενημερωμένης λίστας
```

```
myList[1] = 20 print(myList)
#Θα λάβουμε [1, 20, 3, 4, 5, Γεια]
#append ένα νέο στοιχείο στο myList και εκτύπωση της ενημερωμένης λίστας
myList.append("Πώς είσαι") print(myList)
#Θα λάβουμε [1, 20, 3, 4, 5, 'Γεια', 'Πώς είσαι']
#αφαιρούμε το έκτο στοιχείο από το myList και τυπώνουμε την ενημερωμένη λίστα
del myList[5] print(myList)
#Θα λάβουμε [1, 20, 3, 4, 5, "Πώς είσαι"]
```

30.1.1 Λεξικό - Dictionary

Το λεξικό είναι μια συλλογή “ζευγαριών» τιμών. Για παράδειγμα, αν θέλουμε να αποθηκεύσουμε το όνομα χρήστη και τις ηλικίες 5 χρηστών, μπορούμε να τα αποθηκεύσουμε σε ένα λεξικό.

Για να δηλώσουμε ένα λεξικό, γράφουμε:

dictName= {κλειδί λεξικού: δεδομένα}, με την προϋπόθεση ότι τα κλειδιά λεξικού πρέπει να είναι μοναδικά (μέσα σε ένα λεξικό).

Δηλαδή, δεν πρέπει να δηλώσουμε ένα λεξικό σαν αυτό

myDictionary = {"Peter":38, "John":51, "Peter":13} #Είναι λάθος

Αυτό συμβαίνει επειδή το "Peter" χρησιμοποιείται ως κλειδί λεξικού δύο φορές. Σημειώστε ότι χρησιμοποιούμε αγκύλες { } κατά τη δήλωση ενός λεξικού. Πολλαπλά ζεύγη χωρίζονται με κόμμα.

Παράδειγμα:

```
userNameAndAge = {"Peter":38, "John":51, "Alex":13,
"Alvin":"Μη Διαθέσιμο"}
```

Μπορούμε επίσης να δηλώσουμε ένα λεξικό χρησιμοποιώντας τη μέθοδο dict( ). Για να δηλώσουμε το παραπάνω λεξικό usernameAndAge, γράφουμε:

```
userNameAndAge = dict(Peter = 38, John = 51, Alex = 13,
Alvin
= "Μη διαθέσιμη")
```

Όταν χρησιμοποιούμε αυτήν τη μέθοδο για να δηλώσουμε ένα λεξικό, χρησιμοποιούμε παρενθέσεις ( )

αντί για αγκύλες { } και δεν βάζουμε εισαγωγικά για τα κλειδιά του λεξικού.

Για να αποκτήσουμε πρόσβαση σε μεμονωμένα στοιχεία στο λεξικό, χρησιμοποιούμε το κλειδί λεξικού, το οποίο

είναι η πρώτη τιμή στο ζεύγος {dictionary key : data}. Για παράδειγμα, για να πάρουμε την ηλικία του John, γράφουμε userNameAndAge["John"]. Θα πάρουμε την τιμή 51.

Για να τροποποιήσουμε στοιχεία σε ένα λεξικό, γράφουμε

λεξικόΌνομα [λεξικό κλειδί στοιχείου προς τροποποίηση] = νέα δεδομένα.

Για παράδειγμα, για να τροποποιήσουμε το ζεύγος "John":51, γράφουμε

userNameAndAge["John"] = 21. Το λεξικό μας γίνεται τώρα

userNameAndAge = {"Peter":38, "John":21, "Alex":13,

"Alvin":"Not Available"}.

Μπορούμε επίσης να δηλώσουμε ένα λεξικό χωρίς να του εκχωρήσουμε αρχικές τιμές.

Απλώς γράφουμε λεξικό Όνομα = { }. Αυτό που έχουμε τώρα είναι ένα κενό λεξικό χωρίς στοιχεία.

Για να προσθέσουμε στοιχεία σε ένα λεξικό, γράφουμε λεξικόΌνομα[κλειδί λεξικού] = δεδομένα.

Για παράδειγμα, αν θέλουμε να προσθέσουμε το "Joe":40 στο λεξικό μας, γράφουμε userNameAndAge["Joe"] = 40. Το λεξικό μας γίνεται τώρα

```
userNameAndAge = {"Peter":38, "John":21, "Alex":13,
"Alvin":"Not Available", "Joe":40}
```

Για να διαγράψουμε στοιχεία από ένα λεξικό, γράφουμε

del Όνομα λεξικού[κλειδί λεξικού].

Για παράδειγμα, για να αφαιρέσουμε το ζεύγος

"Alex":13, γράφουμε del userNameAndAge["Alex"]. Το λεξικό μας

τώρα γίνεται userNameAndAge = {"Peter":38, "John":21,

"Alvin":"Not Available", "Joe":40}

Επιπρόσθετα παραδείγματα σε μορφή προγράμματος της Python γράψτε και τρέξτε τον κώδικα σε ένα αρχειάκι:

```
#Δήλωση του λεξικού, τα κλειδιά και τα δεδομένα μπορεί να
είναι άλλου τύπου
myDict = {"One":1.35, 2.5:"Two Point Five", 3:"+", 7.9:2}
#εκτύπωση του λεξικού
print(myDict)
#Θα πάρουμε{'One': 1.35, 2.5: 'Two Point Five', 3: '+',
7.9:2}
#Τα αντικείμενα μπορεί να παρουσιάζονται με διαφορετική
σειρά #Εκτύπωση της τιμής με κλειδί = "One".
print(myDict["One"]) #Θα πάρουμε 1.35
#Εκτύπωση της τιμής με κλειδί = 7.9.
print(myDict[7.9])
```

```
#Θα πάρουμε το 2
#Αλλαγή της τιμής με κλειδί = 2.5 και επανεκτύπωση του
νέου λεξικού
myDict[2.5] = "Two and a Half" print(myDict)
#θα έχουμε {'One': 1.35, 2.5: 'Two and a Half', 3: '+',
7.9:2} # Προσθήκη ενός στοιχείου και εκτύπωση του νέου
λεξικού myDict["New item"] = "I’m new"
print(myDict)
#θα πάρουμε {'One': 1.35, 2.5: 'Two and a Half', 3: '+',
7.9:2, 'New item':
'I’m new'}
#διαγραφή της τιμής με κλειδί = "One" και εκτύπωση του
νέου λεξικού
del myDict["One"] print(myDict)
#Θα πάρουμε {2.5: 'Two and a Half', 3: '+', 7.9: 2, 'New
item': 'I'm new'}
```

30.1.2 Μετατροπή/Αλλαγή τύπου δεδομένων (Casting)

Στην Python, το "casting" αναφέρεται στη διαδικασία μετατροπής μιας τιμής από έναν τύπο δεδομένων σε έναν άλλο. Αυτό είναι μερικές φορές απαραίτητο όταν θέλουμε να εκτελέσουμε λειτουργίες ή συγκρίσεις μεταξύ τιμών διαφορετικών τύπων δεδομένων.

Η Python παρέχει πολλές ενσωματωμένες λειτουργίες για casting, όπως:

```
int(): για τη μετατροπή μιας τιμής σε ακέραιο
x = int(3.14) # το x θα γίνει 3 y = int('123') # το y θα
γίνει 123
float(): μετατροπή σε float
x = float(5) # το x θα γίνει 5.0
y = float('3.14') # το y θα γίνει 3.14

str(): (): για τη μετατροπή μιας τιμής σε string x =
str(123) # το x θα γίνει '123'
y = str(3.14) # το y θα γίνει '3.14'
```

```
bool(): για μετατροπή σε boolean x = bool(0) # η x θα
είναι False
y = bool('') # η y θα είναι False z = bool(42) # η z θα
είναι True
```

30.1.3 Εύρεση του τύπου δεδομένων

Μπορείτε να βρείτε τον τύπο δεδομένων οποιουδήποτε αντικειμένου χρησιμοποιώντας τη συνάρτηση type():

Παράδειγμα

Εκτυπώστε τον τύπο δεδομένων της μεταβλητής x:

```
x = 5
print(type(x))
```

30.1.4 Καθορισμός τύπου μεταβλητής (Casting)

Μπορεί να υπάρχουν φορές που θέλουμε να καθορίσουμε τον τύπο σε μια μεταβλητή. Αυτό μπορεί να γίνει με casting. Η Python είναι μια αντικειμενοστραφής γλώσσα και ως εκ τούτου χρησιμοποιεί κλάσεις για να ορίσει τύπους δεδομένων, συμπεριλαμβανομένων και των πρωτόγονων τύπων της.

Επομένως, το casting γίνεται χρησιμοποιώντας συναρτήσεις κατασκευαστή (constructor):

int() - κατασκευάζει έναν ακέραιο αριθμό από έναν ακέραιο literal, έναν

float literal (αφαιρώντας όλα τα δεκαδικά) ή έναν string literal (με την προϋπόθεση ότι η συμβολοσειρά αντιπροσωπεύει έναν ακέραιο αριθμό)

float() - κατασκευάζει έναν αριθμό float από έναν ακέραιο literal, ένα float literal ή ένα string literal (με την προϋπόθεση ότι η συμβολοσειρά αντιπροσωπεύει έναν float ή έναν ακέραιο)

str() - κατασκευάζει μια συμβολοσειρά από μια μεγάλη ποικιλία τύπων δεδομένων, συμπεριλαμβανομένων συμβολοσειρών, literal ακεραίων και literal float αριθμών

Παράδειγμα

Παραδείγματα μετατροπής μεταβλητών σε άλλο τύπο (casting):

```
x = 5
y = 3.14
z = '10'


# Casting από int σε float
a = float(x) # η a θα γίνει 5.0

# Casting από float σε int b = int(y) # η b θα γίνει 3

# Casting από str σε int
c = int(z) # η c θα γίνει 10

# Casting από int σε str
d = str(x) # η d θα γίνει '5'

# Casting από float σε str
e = str(y) # η θα γίνει '3.14'

# Casting από bool σε int
f = int(True) # η f θα γίνει 1

# Casting από int σε bool
g = bool(0) # η g θα γίνει False
```

30.1.5 Αλληλεπίδραση με το χρήστη

Τώρα που καλύψαμε τα βασικά των μεταβλητών, ας γράψουμε ένα πρόγραμμα για να τα χρησιμοποιήσουμε. Θα ξαναεπισκεφτούμε το πρόγραμμα "Hello World", αλλά αυτή τη φορά θα το κάνουμε διαδραστικό. Αντί να πούμε μόνο ένα γεια στον κόσμο, θέλουμε ο κόσμος να μάθει τα ονόματα και τις ηλικίες μας επίσης. Για να το κάνουμε αυτό, το πρόγραμμά μας πρέπει να μπορεί να μας προτρέπει για πληροφορίες και να τις εμφανίζει στην οθόνη.

Υπάρχουν δύο συναρτήσεις που μπορούν να το κάνουν αυτό:

Η input() και η print().

Προς το παρόν, ας πληκτρολογήσουμε το ακόλουθο πρόγραμμα στο IDLE. Αποθηκεύστε το και εκτελέστε το.

```
myName = input ("Παρακαλώ εισάγετε το όνομά σας: ")
myAge = input ("Ποια είναι ηλικία σας: ")
print ("Γειά σου κόσμε, το όνομά μου είναι ", myName,
"και είμαι ", myAge, "χρονών
```

30.1.6 Συνάρτηση Input()

Στο παραπάνω παράδειγμα, χρησιμοποιήσαμε τη συνάρτηση input() δύο φορές για να λάβουμε από το χρήστη το όνομα και την ηλικία του.

myName = input ("Παρακαλώ εισάγετε το όνομά σας: ")

Η συμβολοσειρά (string) "Παρακαλώ εισάγετε το όνομά σας: " είναι η προτροπή που θα εμφανίζεται στην οθόνη για να δώσει οδηγίες στον

χρήστη. Εδώ, χρησιμοποιήσαμε μια απλή συμβολοσειρά ως προτροπή. Μπορούμε επίσης να χρησιμοποιήσουμε τον τελεστή % ή τη μέθοδο format() για τη μορφοποίηση της συμβολοσειράς εισόδου.

Αφού εμφανιστεί η ερώτηση στην οθόνη, θα περιμένουμε την είσοδο του χρήστη.

Στη συνέχεια, αυτές οι πληροφορίες αποθηκεύονται ως συμβολοσειρά

στη μεταβλητή myName. Η επόμενη ερώτηση προτρέπει τον χρήστη για την ηλικία του και αποθηκεύει τις πληροφορίες ως συμβολοσειρά στη μεταβλητή myAge.

Έτσι λειτουργεί η συνάρτηση input().

Όπως αναφέρθηκε παραπάνω, εκτός από τη χρήση μιας απλής συμβολοσειράς ως προτροπής, εμείς μπορούμε επίσης να χρησιμοποιήσουμε το σύμβολο «%» ή τη μέθοδο format() για να εμφανίσουμε τις μεταβλητές στο μήνυμα (στην εκτύπωση).

Για παράδειγμα, μπορούμε να αλλάξουμε τη δεύτερη ερώτηση παραπάνω από

myAge = input ("Ποια είναι η ηλικία σας: ")

σε

myAge = input("Γεια σου %s, ποια είναι η ηλικία σας: " %(myName))

θα δούμε:

Γεια σου Γιάννης, ποια είναι η ηλικία σας:

30.1.7 Συνάρτηση Print()

Τώρα, ας προχωρήσουμε στη συνάρτηση print(). Η συνάρτηση print() χρησιμοποιείται για να εμφανίζουμε πληροφορίες στους χρήστες.

Δέχεται μηδέν ή περισσότερες εκφράσεις ως ορίσματα, χωρισμένα με κόμμα.

1ος τρόπος εκτύπωσης

```
# Εκτύπωση  συμβολοσειράς
print("Hello, World!")
# έξοδος: Hello, World!

# Εκτύπωση μεταβλητής
x = 42
print(x)
# έξοδος: 42
```

2ος τρόπος εκτύπωσης

Η μορφοποίηση συμβολοσειρών είναι μια ισχυρή τεχνική που μας επιτρέπει να εισάγουμε τιμές σε μια συμβολοσειρά. Στην Python, μπορούμε να χρησιμοποιούμε τον τελεστή '%' για να μορφοποιούμε μια συμβολοσειρά. Αυτός ο τελεστής παίρνει μια συμβολοσειρά στην αριστερή πλευρά και μία ή περισσότερες τιμές στη δεξιά πλευρά, διαχωρισμένες με κόμματα.

```
# Εκτύπωση formatted string
name = "Alice"
age = 25
print("%s is %d years old." % (name, age))
# Έξοδος: Alice is 25 years old.

# Εκτύπωση formatted string με μορφοποίηση δεκαδικού
price = 19.99
print("The price is $%.2f." % price)
# Έξοδος: The price is $19.99.
```

3ος τρόπος εκτύπωσης

Η μέθοδος «str.format()» είναι ένας νεότερος, πιο ευέλικτος τρόπος μορφοποίησης συμβολοσειρών στην Python. Αυτή η μέθοδος μας επιτρέπει να εισάγουμε τιμές σε μια συμβολοσειρά χρησιμοποιώντας άγκιστρα "{}" και στη συνέχεια να μεταβιβάσουμε αυτές τις τιμές ως ορίσματα στη μέθοδο "format()".

```
# Εκτύπωση με το 'str.format()'
name = "Bob"
age = 30
print("{} is {} years old.".format(name, age))
# έξοδος: Bob is 30 years old.
```

```
# Εκτύπωση με προκαθορισμένες μεταβλητές
product = "Python Course"
price = 99.99
print("{name} costs ${price:.2f}.".format(name=product,
price=price))
# Έξοδος: Python Course costs $99.99.
```

4ος τρόπος εκτύπωσης

Η σύνταξη «f-string» είναι ένας ακόμα πιο καινούριος, ακόμη πιο συνοπτικός τρόπος μορφοποίησης συμβολοσειρών στην Python. Αυτή η σύνταξη μας επιτρέπει να ενσωματώνουμε εκφράσεις κυριολεκτικά απευθείας σε συμβολοσειρές, προσθέτοντας το πρόθεμα «f».

```
# Εκτύπωση με f-string
name = "Carol"
age = 35
print(f"{name} is {age} years old.")
# Έξοδος: Carol is 35 years old.

# Εκτυπώνοντας f-string με μορφοποιημένο δεκαδικό
price = 29.95
tax_rate = 0.08
print(f"The total price is ${price * (1 +
tax_rate):.2f}.")
# έξοδος: The total price is $32.34.
```

5ος τρόπος εκτύπωσης

Εκτός από την εκτύπωση στην κονσόλα, μπορούμε επίσης να εκτυπώσουμε σε ένα αρχείο ή σε μια ροή στην Python. Αυτό μας είναι

χρήσιμο όταν θέλουμε να αποθηκεύσουμε το αποτέλεσμα για μεταγενέστερη ανάλυση ή να το μοιραστούμε με άλλους. Για να το κάνουμε αυτό, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση print με το όρισμα λέξης κλειδιού «file».

```
# Εκτύπωση σε αρχείο
with open('output.txt', 'w') as f:
    # Εγγραφή απευθείας στο αρχείο
    print("This line will be written to a file.", file=f)

# Εκτύπωση σε ένα stream
import sys
print("This line will be written to a stream.",
file=sys.stderr)
```

Τριπλά εισαγωγικά

Σε ορισμένες περιπτώσεις, μπορεί να θέλουμε να εμφανίσουμε ένα μεγάλο μήνυμα χρησιμοποιώντας την print(). Για να το κάνουμε αυτό, μπορούμε να χρησιμοποιήσουμε το σύμβολο τριπλών εισαγωγικού (''' ή """) για να τυπώσουμε το μήνυμά μας σε πολλές γραμμές. Για παράδειγμα:

```
print ('''Γεια σου κόσμε.
Το όνομά μου είναι Γιάννης και είμαι 20 χρονών.''')
# θα μας δώσει:
«Γεια σου κόσμε.
Το όνομά μου είναι Γιάννης και είμαι 20 χρονών.»
```

Αυτό βοηθά στην αναγνωσιμότητα του κώδικα και του τυπωμένου μηνύματός μας.

30.1.9 Χαρακτήρες διαφυγής

Μερικές φορές μπορεί επίσης να χρειαστεί να εκτυπώσουμε κάποιους ειδικούς «μη εκτυπώσιμους» χαρακτήρες όπως ένα χαρακτήρα tab ή ένα χαρακτήρα αλλαγής γραμμής. Σε αυτήν την περίπτωση, πρέπει να χρησιμοποιήσουμε την "\" (αντίθετη κάθετο, backslash).

Για παράδειγμα, για να εκτυπώσουμε ένα tab, πληκτρολογούμε τον χαρακτήρα αντίθετης καθέτου πριν από το γράμμα t, όπως αυτό: \t.

Χωρίς τον χαρακτήρα \, το γράμμα t θα εκτυπωθεί κανονικά.

Με αυτό, τυπώνεται ένα tab. Ως εκ τούτου, αν πληκτρολογήσουμε

print ("Γεια σου\tκόσμε"), θα δούμε:

Γειά σου κόσμε

Άλλες κοινές χρήσεις του χαρακτήρα αντίθετης καθέτου φαίνονται παρακάτω:

\n (Εκτυπώνει μια νέα γραμμή)

Δίνοντας: print (Γεια σου\nκόσμε'), θα πάρουμε:

Γειά σου

κόσμε

\\ (Εκτυπώνει τον ίδιο τον χαρακτήρα κάθετο)

Δίνοντας: print('\\'), θα πάρουμε:

\
\" (Εκτυπώνει διπλό εισαγωγικό, έτσι ώστε το διπλό εισαγωγικό να μην σηματοδοτεί το τέλος της συμβολοσειράς)

Δίνοντας: print (“Ο Σωκράτης είπε: \”Εν οίδα ότι ουδέν οίδα\”")

Παίρνουμε: Ο Σωκράτης είπε: ”Εν οίδα ότι ουδέν οίδα”

\' (Χρησιμοποιήστε μονά και διπλά εισαγωγικά ώστε να μπορείτε να εκτυπώσετε τα εισαγωγικά που θέλετε)

Δίνοντας: print (“Ο Σωκράτης είπε: \’Εν οίδα ότι ουδέν οίδα\’")

Παίρνουμε: Ο Σωκράτης είπε: ‘Εν οίδα ότι ουδέν οίδα’

6ος τρόπος εκτύπωσης

Εάν δεν θέλουμε οι χαρακτήρες που προηγούνται του χαρακτήρα \ να

ερμηνεύονται ως ειδικοί χαρακτήρες, δηλαδή να τυπώσουμε ακριβώς αυτό που βλέπουμε στην print(), μπορούμε να χρησιμοποιήσουμε ακατέργαστες συμβολοσειρές (raw strings) προσθέτοντας ένα r πριν από το πρώτα εισαγωγικά:

Για παράδειγμα, εάν δεν θέλουμε το \t να ερμηνεύεται ως tab, πρέπει να πληκτρολογήσουμε: print (r' Γεια σου\tκόσμε ').

Έτσι θα έχουμε «Γεια σου\tκόσμε» ως έξοδο.

30.1.10 Λήψη Αποφάσεων – if, for, while

Φτάσαμε σε ένα πιο ενδιαφέρον μέρος της μάθησης, τη λήψη αποφάσεων. Αυτό θα μας επιτρέπει να φτιάχνουμε πρόγραμμα πιοέξυπνο, ικανό να κάνει επιλογές και αποφάσεις. Θα εξετάσουμε τη δήλωση if, τον βρόχο for και τον βρόχο while.

Αυτές οι δομές είναι γνωστές ως εργαλεία ελέγχου ροής.

Ελέγχουν τη ροή του προγράμματος. Επιπλέον, θα εξετάσουμε επίσης τα try και except τα οποία καθορίζουν τι πρέπει να κάνει το πρόγραμμαόταν παρουσιαστεί σφάλμα.

Παραδείγματα

```
a = 200
b = 33
if b > a:
print("το b είναι μεγαλύτερο από το a")
elif a == b:
print("το a και το b είναι ίσα")
else:
print("το a είναι μεγαλύτερο από το b")
```

Μπορούμε επίσης να έχουμε else χωρίς να έχουμε το elif (όταν έχουμε μόνο δύο περιπτώσεις για επιλογή):

2.

```
a = 200
b = 33
if b > a:
print("το b είναι μεγαλύτερο από το a")
else:
print("το b δεν είναι μεγαλύτερο από το a")
```

Inline if

Μια τέτοια δήλωση είναι μια πιο απλή μορφή μιας δήλωσης if και είναι πιο βολική, αν θέλουμε να γράψουμε κάτι απλό.

Η σύνταξη είναι:

```
if a > b: print("το a είναι μεγαλύτερο από το b ")
```

Τριαδικός τελεστής (ternary operator) – Shorthand if

Αν έχουμε μόνο μια πρόταση για εκτέλεση, μπορούμε να τη βάλουμε στην ίδια γραμμή με το if:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else
print("B")
```

30.1.11 Εμφωλιασμένο if (nested – if)

```
x = 41
if x > 10:
      print("Πάνω από 10,")
      if x > 20:
            print("and also above 20!")
else:
print("αλλά όχι πάνω από 20.")
```

30.1.12 Βρόχος for

Παραδείγματα

```
for i in range(5):
      print(i)
# Έξοδος: 0 1 2 3 4 (το καθένα σε άλλη γραμμή)
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

30.1.13 Εμφωλιασμένα - Nested for loops

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Έξοδος:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Loop σε λεξικό

```
age = {'Peter': 5, 'John':7}
for i in age:
print(i)
```

Εάν θέλουμε να λάβουμε και το κλειδί του λεξικού και τα δεδομένα,

μπορούμε να το κάνουμε ως εξής:

```
age = {'Peter': 5, 'John':7}
for i in age:
      print("Name = %s, Age = %d" %(i, age[i]))
```

30.1.14 While loop

Με το while μπορούμε να εκτελέσουμε εντολές, όσο ισχύει μια συνθήκη.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Δηλώσεις break, continue, pass

Το break μας βγάζει από το loop αν εκπληρωθεί μια συνθήκη:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
i += 1
```

Το continue σταματάει την παρούσα επανάληψη και συνεχίζει με την επόμενη:

```
i = 0
while i < 6:
  i += 1
if i == 3:
    continue
print(i)
```

Το pass δεν κάνει τίποτα και χρησιμοποιείται πολλές φορές για να συμπληρώσουμε κάτι αργότερα (όμως ο κώδικάς μας να τρέχει).

```
while i < 6:
  i += 1
if i == 3:
    pass
print(i)
```

30.1.15 Συναρτήσεις

Οι συναρτήσεις είναι έτοιμα τμήματα κώδικα τα οποία εκτελούν μια συγκεκριμένη εργασία.

Μπορούμε να καλέσουμε μια συνάρτηση απλά πληκτρολογώντας το όνομα της συνάρτησης ή χρησιμοποιώντας τον συμβολισμό με την τελεία «.» (dot notation).

Για παράδειγμα, για να χρησιμοποιήσουμε τη συνάρτηση print() για την εμφάνιση κειμένου στην οθόνη, την καλούμε πληκτρολογώντας

```
print("Γειά σου κόσμε")
```

όπου print είναι το όνομα της συνάρτησης και

"Γειά σου κόσμε " είναι το όρισμα.

Χρησιμοποιώντας τον συμβολισμό με τελεία (dot notation), για να καλέσουμε τη συνάρτηση replace() για τη διαχείριση συμβολοσειρών κειμένου, πρέπει να πληκτρολογήσουμε

```
newString = "Γειά σου κόσμε".replace("κόσμε", "σύμπαν")
```

όπου “replace” είναι το όνομα της συνάρτησης και τα "κόσμε" και "σύμπαν" τα ορίσματα. Η συμβολοσειρά πριν από την τελεία (δηλαδή η "Γειά σου κόσμε") είναι η συμβολοσειρά που θα επηρεαστεί.

Μπορούμε να ορίσουμε τις δικές μας συναρτήσεις στην Python και να τις χρησιμοποιήσουμε ξανά και ξανά μέσα σε ένα πρόγραμμα.
Η πρώτη γραμμή του ορισμού μιας συνάρτησης είναι η επικεφαλίδα της συνάρτησης. Η επικεφαλίδα ξεκινάει με τη δεσμευμένη λέξη def ακολουθούμενη από το όνομα της συνάρτησης, ένα ζευγάρι

παρενθέσεων που προαιρετικά περικλείει μια διαχωριζόμενη με κόμματα λίστα παραμέτρων (parameters) και τελειώνει με μια άνω κάτω τελεία. Κάτω από την επικεφαλίδα ακολουθεί το σώμα της συνάρτησης (οι εντολές της συνάρτησης) σε εσοχή.

Τα ονόματα των συναρτήσεων δίνονται με βάση τους ίδιους κανόνες που είδαμε και για τα ονόματα των μεταβλητών. Καλό είναι να δίνουμε ονόματα σχετικά με τη λειτουργία που επιτελούν οι συναρτήσεις. Οι συναρτήσεις ορίζονται συνήθως στην αρχή των προγραμμάτων. Είναι προφανές όμως ότι μια συνάρτηση πρέπει να οριστεί πρώτα πριν χρησιμοποιηθεί (κληθεί). Κατά την κλήση μιας συνάρτησης εκτελούνται οι εντολές που περιέχονται στο σώμα της.

Μπορούμε να καλούμε μία συνάρτηση όσες φορές απαιτούνται για τη λύση του προβλήματος που αντιμετωπίζει το πρόγραμμά μας.

Η σύνταξη λοιπόν για τον ορισμό μιας συνάρτησης είναι η εξής:

```
def όνομαΣυνάρτησης(λίστα παραμέτρων):
κώδικας που περιγράφει τι πρέπει να κάνει η συνάρτηση
return [έκφραση]
```

Υπάρχουν δύο λέξεις-κλειδιά εδώ, def και return.

30.1.16 Παραδείγματα Συναρτήσεων

Υπολογισμός πρώτου αριθμού

```
def protos_ar(ar_gia_elegxo):
  for x in range(2, ar_gia_elegxo):
    if (ar_gia_elegxo % x == 0):
      return False
    else:
      return True
```

```
apantisi = protos_ar(11765)
print(apantisi)
```

Φτιάξτε μια συνάρτηση που δέχεται ως όρισμα έναν αριθμό και επιστρέφει την απόλυτη τιμή του:

Παράδειγμα – Υπολογισμός απόλυτης τιμής

```
def absValue (x):
if x < 0:
return -x
else:
return x

print(absValue(4))
print(absValue(-4))
```

Παράδειγμα – Υπολογισμός αθροίσματος ή συνένωσης

```
def add(x,y):
return x + y

print(add(9,7))
print(add("Γειά σου ", "κόσμε"))
```

30.1.17 Εμβέλεια μεταβλητών

Μια σημαντική έννοια που πρέπει να κατανοήσουμε κατά τον ορισμό μιας συνάρτησης είναι η έννοια της εμβέλειας των μεταβλητών.

Οι μεταβλητές που ορίζονται μέσα σε μια συνάρτηση αντιμετωπίζονται διαφορετικά από τις μεταβλητές που ορίζονται έξω.

Υπάρχουν δύο βασικές διαφορές.

Πρώτον, οποιαδήποτε μεταβλητή δηλώνεται μέσα σε μια συνάρτηση είναι προσβάσιμη μόνο εντός της συνάρτησης.

Αυτές είναι γνωστές και ως τοπικές μεταβλητές.

Οποιαδήποτε μεταβλητή δηλώνεται εκτός μιας συνάρτησης είναι γνωστή ως καθολική μεταβλητή και είναι προσβάσιμη απ' οπουδήποτε στο πρόγραμμα.

Για να το καταλάβουμε, ας δούμε τον παρακάτω κώδικα:

```
gloVar = "Καθολική μεταβλητή"
def myFunction():
print("\nΜΕΣΑ ΣΤΗ ΣΥΝΑΡΤΗΣΗ")
#Οι καθολικές μεταβλητές ειναι ορατές μέσα σε μια
συνάρτηση
print (gloVar)
#Δήλωση τοπικής μεταβλητής
locVar = "Τοπική μεταβλητή"
print(locVar)
'''
Καλούμε τη συνάρτηση παρακάτω
Παρατηρήστε ότι η myFunction() δεν έχει παραμέτρους.
Έτσι την καλούμε με κενές παρενθέσεις.
'''
myFunction()

print("\nΕΞΩ ΑΠΟ ΤΗ ΣΥΝΑΡΤΗΣΗ")

#Οι καθολικές μεταβλητές είναι ορατές έξω από μια
συνάρτηση
print(gloVar)

# Οι τοπικές μεταβλητές ΔΕΝ είναι ορατές έξω από μια
συνάρτηση
# Έτσι, στο τέλος του προγράμματος θα πάρουμε μήνυμα
λάθους:
# NameError: name 'locVar' is not defined. Did you
mean: 'gloVar'?
print(locVar)
```

Τι γίνεται στην περίπτωση που η τοπική μεταβλητή έχει το ίδιο όνομα με την καθολική;

Ας δούμε ένα παράδειγμα:

```
onoma = "Ελένη"
def pesGeia():
print("Γειά " + onoma)

def alloOnoma(alloOnoma):
onoma = alloOnoma

pesGeia()
alloOnoma("Μαρία")
pesGeia()
```

Στο παραπάνω παράδειγμα, έχουμε τη μεταβλητή onoma σαν καθολική, αλλά και τοπική μεταβλητή. Όταν όμως καλείται η συνάρτηση alloOnoma με το όρισμα «Μαρία», τυπώνεται και πάλι «Ελένη», καθότι:

Όταν η τοπική μεταβλητή έχει το ίδιο όνομα με την καθολική, χρησιμοποιείται η καθολική μεταβλητή κάθε φορά που καλείται η συνάρτηση και όχι η τοπική της.

```
onoma = "Ελένη"
def pesGeia():
print("Γειά " + onoma)

def alloOnoma(alloOnoma):
global onoma
onoma = alloOnoma

pesGeia()
alloOnoma("Μαρία")
pesGeia()
```

```
"""
Αυτή η συνάρτηση δείχνει τη διαφορά
καθολικών και τοπικών μεταβλητών,
```

```
όταν τις καλούμε
"""
onoma = "Ελένη"
def pesGeia():
print("Γειά " + onoma)

def alloOnoma(alloOnoma):
global onoma
onoma = alloOnoma

pesGeia()
alloOnoma("Μαρία")
pesGeia()
```

30.1.18 Μεταβλητός αριθμός ορισμάτων (*)

Εκτός από τις προεπιλεγμένες τιμές για τις παραμέτρους, η Python μας επιτρέπει επίσης να έχουμε μεταβλητό αριθμό ορισμάτων σε μια συνάρτηση.

Αυτό είναι πολύ χρήσιμο εάν εμείς δεν γνωρίζουμε εκ των προτέρων τον αριθμό των ορισμάτων που έχει μια συνάρτηση.

Για παράδειγμα, μπορεί να έχουμε μια συνάρτηση που προσθέτει μια σειρά αριθμών, αλλά δεν ξέρουμε πόσοι αριθμοί υπάρχουν εκ των προτέρων. Σε περιπτώσεις όπως αυτή, μπορούμε να χρησιμοποιήσουμε το σύμβολο «*».

Παράδειγμα:

```
Def addNumbers(*num):
sum = 0
for i in num:
sum = sum + i
print(sum)
```

Αν θέλουμε να περάσουμε σε μια λίστα ορισμάτων μεταβλητού μήκους με λέξεις-κλειδιά στη συνάρτηση, μπορούμε να χρησιμοποιήσουμε διπλό αστερίσκο.

Παράδειγμα:

```
def printMemberAge(**age):
for i, j age.items():
print("Name = %s, Age = %s" %(i, j))
```

Αυτή η συνάρτηση έχει μια παράμετρο που ονομάζεται age (ηλικία). Οι διπλοί αστερίσκοι δηλώνουν ότι αυτή η παράμετρος αποθηκεύει μια λίστα ορισμάτων μεταβλητού μήκους με λέξεις-κλειδιά, η οποία είναι ουσιαστικά ένα λεξικό.

Στη συνέχεια, ο βρόχος for, περνά μέσα από το όρισμα και εκτυπώνει τις τιμές.

Για να καλέσουμε τη συνάρτηση, γράφουμε:

```
printMemberAge(Peter = 5, John = 7)
```

θα πάρουμε:

```
Name = Peter, Age = 5
Name = John, Age = 7
```

Αν γράψουμε:

```
printMemberAge(Peter = 5, John = 7, Yvonne = 10)

# θα πάρουμε:
Name = Peter, Age = 5
Name = John, Age = 7
Name = Yvonne, Age = 10
```

30.1.19 Ασκήσεις

1. Γράψτε ένα πρόγραμμα για να εκτυπώνεται το παρακάτω σχήμα με αστεράκια

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

2. Γράψτε ένα προγραμματάκι το οποίο δέχεται μια λέξη από τον χρήστη και την αντιστρέφει

3. Γράψτε ένα πρόγραμμα το οποίο τυπώνει όλους τους αριθμούς από το 1 μέχρι το 6, εκτός από το 3 και το 6.Σημείωση: Χρησιμοποιήστε το 'continue'.
Αναμενόμενη έξοδος : 0 1 2 4 5

4. Γράψτε ένα μια συνάρτηση σε Python η οποία πολλαπλασιάζει όλους τους αριθμούς της παρακάτω λίστας.
Λίστα : (8, 2, 3, -1, 7)
Αναμενόμενη έξοδος : -336

5. Γράψτε μια συνάρτηση για να ελέγχετε αν ένας αριθμός ανήκει σε ένα εύρος τιμών.

6. Γράψτε μια συνάρτηση για να ελέγξετε εάν ένας αριθμός είναι "τέλειος" ή όχι.

Σύμφωνα με τη Wikipedia : Στη θεωρία αριθμών, ένας τέλειος αριθμός είναι ένας θετικός ακέραιος αριθμός που ισούται με το άθροισμα των θετικών διαιρετών του, δηλαδή το άθροισμα των θετικών του διαιρετών εξαιρουμένου του ίδιου του αριθμού (γνωστό και ως άθροισμα υποπολλαπλασιασμού του). Ισοδύναμα, τέλειος αριθμός είναι ένας

αριθμός που είναι το ήμισυ του αθροίσματος όλων των θετικών διαιρετών του (συμπεριλαμβανομένου του εαυτού του).

Παράδειγμα : Ο πρώτος τέλειος αριθμός είναι το 6, επειδή 1, 2 και 3 είναι οι σωστοί θετικοί διαιρέτες του και 1 + 2 + 3 = 6. Ισοδύναμα, ο αριθμός 6 είναι ίσος με το ήμισυ του αθροίσματος όλων των θετικών του διαιρετών: (1 + 2 + 3 + 6) / 2 = 6.

Ο επόμενος τέλειος αριθμός είναι 28 = 1 + 2 + 4 + 7 + 14. Αυτό ακολουθείται από τους τέλειους αριθμούς 496 και 8128.