31.ΟΡΙΣΜΑΤΑ – ΣΥΝΑΡΤΗΣΕΙΣ – TRY, EXCEPT – ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ

31.0.1 Λύσεις των προηγούμενων ασκήσεων

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

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

Λύση

```
n=5;
for i in range(n):
    for j in range(i):
        print ('* ', end="")
    print('')

for i in range(n,0,-1):
    for j in range(i):
        print('* ', end="")
    print('')
```

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

```
word = input("Input a word to reverse: ")
'''
for char in range(len(word) - 1, -1, -1):
  print(word[char], end="")
print("\n")
'''
print(word[::-1], end="")
print("\n")
```

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

```
for x in range(6):
    if (x == 3 or x==6):
        continue
    print(x,end=' ')
print("\n")
```

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

```
def multiply(numbers):
    total = 1
    for x in numbers:
        total *= x
    return total
print(multiply((8, 2, 3, -1, 7)))
```

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

```
def test_range(n):
    if n in range(3,9):
        print( " %s is in the range"%str(n))
    else :
        print("The number is outside the given range.")
test_range(5)
```

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 ο 28 = 1 + 2 + 4 + 7 + 14. Αυτό ακολουθείται από τους τέλειους αριθμούς 496 και 8128.

```
def perfect_number(n):
    sum = 0
    for x in range(1, n):
        if n % x == 0:
            sum += x
    if sum == n:
        '''
        print(f"Ο αριθμός {n} είναι τέλειος!")
    else:
        print(f"Ο αριθμός {n} ΔΕΝ είναι τέλειος!")
        '''
```

```
    return sum == n
print(perfect_number(8130))
```

31.1.0 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 μια συνάρτηση η οποία ζητάει το όνομα και το επίθετο του χρήστη, την get_formatted_name(), ώστε να χειρίζεται και μεσαία ονόματα.

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.

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
```

31.2.0 Δημιουργία συναρτήσεων και βρόχος 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 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:;
```

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 = []
usernames = [item for item in input("Εισάγετε \
                τα ονόματα: ").split()]
#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.

31.3.0 Αντίγραφο ασφαλείας λίστας

Πολλές φορές χρειάζεται να χρησιμοποιούμε ένα αντίγραφο της λίστας που έχουμε, καθότι ξαναχρειαζόμαστε την αρχική λίστα κάπου αλλού στο πρόγραμμά μας.

Ένας τρόπος για να λαμβάνουμε αντίγραφο μιας λίστας είναι με το slice notation. Υπάρχουν αρκετοί άλλοι τρόποι. Ας δούμε ένα παράδειγμα:

```
origi_list = ['κείμενο', 0, 89, 2, 6.7]

# Αντιγραφή λίστας με slicing
new_list = origi_list[:]

# Προσθήκη νέου στοιχείου στη νέα λίστα
new_list.append('αύριο')

# Εκτύπωση και των δύο λιστών
print('Η παλιά λίστα είναι:', origi_list)
print('Η νέα λίστα είναι:', new_list)
```

31.4.0 Δηλώσεις ελέγχου try, except

Η δήλωση ελέγχου try, except είναι μια γνωστή δήλωση στον προγραμματισμό. Αυτή η δήλωση ελέγχει πώς συνεχίζει ένα πρόγραμμα αφού παρουσιαστεί ένα σφάλμα. Η σύνταξη έχει ως εξής:

```
try:
      κάνε κάτι
except:
      κάνε κάτι άλλο όταν παρουσιαστεί ένα σφάλμα
```

Για παράδειγμα, ας δοκιμάσουμε να εκτελέσουμε το παρακάτω πρόγραμμα:

```
try:
    answer = 12/0
    print (answer)
except:
    print ("An error occurred")
```

Αν εκτελέσουμε το πρόγραμμα, θα λάβουμε το μήνυμα "Παρουσιάστηκε σφάλμα".

Αυτό συμβαίνει γιατί όταν το πρόγραμμα προσπαθεί να εκτελέσει την πρόταση answer = 12/0 στο μπλοκ try, παρουσιάζεται σφάλμα αφού δεν μπορούμε να διαιρέσουμε έναν αριθμό με το μηδέν.

Το υπόλοιπο του μπλοκ try αγνοείται και η δήλωση στο μπλοκ except εκτελείται αντ' αυτού.

Αν θέλουμε να εμφανίζονται πιο συγκεκριμένα και σχετικά με το σφάλμα μηνύματα σφάλματος στους χρήστες, μπορούμε να καθορίσουμε τον τύπο σφάλματος μετά τη λέξη-κλειδί except. Ας δούμε τον παρακάτω κώδικα:

```
try:
   userInput1 = int(input("Παρακαλούμε δώστε έναν αριθμό σαν διαιρετέο: "))
    userInput2 = int(input("Παρακαλούμε δώστε ακόμα έναν σαν διαιρέτη του: "))
    answer =userInput1/userInput2
    print ("Η απάντηση είναι ", answer)
    #myFile = open("missing.txt", 'r')
except ValueError:
    print ("Σφάλμα: Δεν δώσατε αριθμό")
except ZeroDivisionError:
    print ("Σφάλμα: Δεν μπορεί να γίνει διαίρεση με το μηδέν")
except Exception as e:
    print ("Άγνωστο σφάλμα: ", e)
```

Η παρακάτω λίστα δείχνει τις διάφορες εξόδους για διαφορετικές εισόδους χρηστών.

Το “>>>” δηλώνει την είσοδο του χρήστη και το “=>” υποδηλώνει την έξοδο.

>>> Εισάγετε έναν αριθμό: m

=> Σφάλμα: Δεν δώσατε αριθμό

Αιτία: Ο χρήστης εισήγαγε μια συμβολοσειρά που δεν μπορεί να μετατραπεί σε έναν ακέραιο. Αυτό είναι ένα ValueError.

Έτσι, εμφανίζεται η δήλωση του μπλοκ except ValueError.

```
>>> Εισάγετε έναν αριθμό: 12
>>> Εισάγετε έναν άλλο αριθμό: 0
```

```
=> Σφάλμα: Δεν μπορεί να γίνει διαίρεση με το μηδέν
```

Αιτία: userInput2 = 0. Εφόσον δεν μπορούμε να διαιρέσουμε έναν αριθμό με το μηδέν, αυτό είναι ένα ZeroDivisionError.

Έτσι, εμφανίζεται η δήλωση του μπλοκ except ZeroDivisionError

```
>>> Εισάγετε έναν αριθμό: 12
>>> Εισάγετε έναν άλλο αριθμό: 3
=> Η απάντηση είναι 4.0
=> Άγνωστο σφάλμα: [Errno 2] Δεν υπάρχει τέτοιο αρχείο ή
κατάλογος:
'missing.txt'
```

Αιτία: Ο χρήστης εισάγει αποδεκτές τιμές και η γραμμή

print ("Η απάντηση είναι ", answer) εκτελείται σωστά.

Ωστόσο, η επόμενη γραμμή εμφανίζει ένα σφάλμα ως το αρχείο

missing.txt δεν βρέθηκε.

Επειδή αυτό δεν είναι ***ValueError*** ή ***ZeroDivisionError***, εκτελείται το τελευταίο μπλοκ except.

Το ***ValueError*** και το ***ZeroDivisionError*** είναι δύο από τους πολλούς τύπους προκαθορισμένων σφαλμάτων της Python.

Το ***ValueError*** εμφανίζεται όταν μια ενσωματωμένη λειτουργία ή συνάρτηση λαμβάνει ένα όρισμα που έχει τον σωστό τύπο αλλά ακατάλληλη τιμή.

Το ***ZeroDivisionError*** εμφανίζεται όταν το πρόγραμμα προσπαθεί να διαιρέσει με το μηδέν.

Άλλα κοινά σφάλματα στην Python περιλαμβάνουν τα:

IOError:

Εμφανίζεται όταν μια λειτουργία I/O (όπως η ενσωματωμένη λειτουργία open()) αποτυγχάνει) για ένα λόγο που σχετίζεται με είσοδο/έξοδο, π.χ. "το αρχείο δεν βρέθηκε".

ImportError:

Εμφανίζεται όταν μια δήλωση import δεν μπορεί να βρει το ζητούμενο module.

IndexError:

Εμφανίζεται όταν ένας δείκτης ακολουθίας (π.χ. συμβολοσειρά, λίστα, πλειάδα) είναι εκτός εύρους.

KeyError::

Εμφανίζεται όταν δεν βρίσκεται ένα κλειδί λεξικού.

NameError:

Εμφανίζεται όταν δεν βρεθεί μια τοπική ή καθολική μεταβλητή.

TypeError:

Εμφανίζεται όταν μια λειτουργία ή συνάρτηση εφαρμόζεται σε ένα αντικείμενο ακατάλληλου τύπου.

Για μια πλήρη λίστα όλων των τύπων σφαλμάτων στην Python, μπορείτε να ανατρέξετε εδώ:

https://docs.python.org/3/library/exceptions.html .

Η Python έρχεται επίσης με προκαθορισμένα μηνύματα σφάλματος για κάθε ένα από τα διαφορετικά είδη σφαλμάτων. Εάν θέλουμε να εμφανίσουμε το μήνυμα, χρησιμοποιούμε τη λέξη-κλειδί “as” μετά τον τύπο σφάλματος.

Για παράδειγμα, για να εμφανίσουμε το προεπιλεγμένο μήνυμα ValueError, γράφουμε:

except ValueError as e:

print (e)

Το e είναι το όνομα της μεταβλητής που έχει εκχωρηθεί στο σφάλμα. Μπορούμε να του δώσουμε κάποιο άλλο όνομα, αλλά είναι κοινή πρακτική η χρήση του e.

Η τελευταία δήλωση except στο πρόγραμμά μας:

except Exception as e:

print ("Unknown error: ", e)

είναι ένα παράδειγμα χρήσης και πάλι προκαθορισμένου μηνύματος σφάλματος. Χρησιμεύει ως ένα τελικό μήνυμα προσπαθώντας να εντοπίσει τυχόν απρόβλεπτα σφάλματα.

Παράδειγμα 1:

```
try:
    lst = ['Γιάννης', 'Μαρία', 'Νίκος']
    print(lst[3])
except Exception as e:
    print(e)
print('Άραγε τυπώθηκα; Ή έγινε κάποιο λάθος;')
```

όπου λαμβάνουμε list out of index error, αφού το τέταρτο στοιχείο που ζητάμε, δεν υπάρχει

Παράδειγμα 2:

Ας δούμε ένα παράδειγμα στο οποίο προσπαθούμε να προσθέσουμε μια συμβολοσειρά με έναν ακέραιο, το οποίο θα μας δώσει ένα σφάλμα. Ζητάμε τον τύπο του σφάλματος για debugging:

```
try:
    "string" + 1
except Exception as e:
    print(e)
    #print(type(e).__name__)
```

31.4.1 Δηλώσεις ελέγχου try, except

Ας δούμε κι άλλα κλασικά try – except errors

Παράδειγμα 3

```
try:
    x = undefined_variable
except Exception as e:
    print(str(e))
```

Εδώ έχουμε μια μη-ορισμένη μεταβλητή η οποία θα προκαλέσει ένα σφάλμα NameError το οποίο θα τυπωθεί.

Σημειολογικά, αυτό είναι το ίδιο με το print(e) το οποίο εσωτερικά, καλεί την str(e) για να βρει μια αντιπροσώπευση του e.

έξοδος:

```
name 'undefined_variable' is not defined
```

31.4.2 Παράδειγμα 4

Εκτύπωση ανίχνευσης της εξαίρεσης

Χρησιμοποιώντας το module traceback, μπορούμε να ανακτήσουμε κα να τυπώσουμε όλο το traceback μιας εξαίρεσης, οδηγώντας σε καλύτερη κατανόηση του τι συνέβη.

```
import traceback
try:
    open("nonexistent_file.txt")
except Exception:
    print(traceback.format_exc())
```

Εδώ, προσπαθούμε να ανοίξουμε ένα ανύπαρκτο αρχείο, το οποίο θα προκαλέσει ένα FileNotFoundError. Το traceback τυπώνεται, παρέχοντάς μας πληροφορίες για το σφάλμα.

Έξοδος:

```
Traceback (most recent call last):
...
FileNotFoundError: [Errno 2] No such file or directory:
'nonexistent_file.txt'
```

31.4.3 Παράδειγμα 5

Εκτύπωση ανίχνευσης στοίβας της εξαίρεσης

Τυπώνοντας την ανίχνευση της στοίβας, πηγαίνουμε σιγά-σιγά προς τα πίσω μέχρι το σημείο στο οποίο κλήθηκε η εξαίρεση. Αυτή η μέθοδος είναι ιδιαίτερα χρήσιμη για την κατανόηση της σειράς των λαθών που οδήγησαν στο σφάλμα, βοηθώντας τον προγραμματιστή να λύσει το λάθος.

```
import traceback
try:
    list_of_numbers = [1, 2, 3]
    print(list_of_numbers[3])
except Exception:
    print(traceback.print_stack())
```

Είναι ένα σφάλμα το οποίο εξετάσαμε παραπάνω και προκαλεί IndexError. Τυπώνεται η ανίχνευση της στοίβας δείχνοντας το μονοπάτι του λάθους.

Έξοδος: None

31.4.4 Παράδειγμα 6

Εκτύπωση ονόματος εξαίρεσης

Printing the exception name involves capturing the exception and using type(e).__name__ to retrieve and display the name of the exception type. This method provides clear information about the kind of error that occurred, such as TypeError, and is useful for logging and debugging purposes.

```
try:
    "string" + 1
except Exception as e:
    print(type(e).__name__)
```

In this code, we attempt to concatenate a string and an integer, which will raise a TypeError. The name of the exception type is printed.

Έξοδος: TypeError

31.4.5 Παράδειγμα 7

Εκτύπωση τύπου εξαίρεσης

Χρησιμοποιούμε τη συνάρτηση type(e) για να ανακτήσουμε και να τυπώσουμε το είδος της εξαίρεσης. Αυτή η μέθοδος παρέχει μια εξήγηση της κατηγορίας του λάθους, βοηθώντας στην αποσφαλμάτωση.

```
try:
    {}["key"]
except Exception as e:
    print(type(e))
```

Προσπαθούμε να προσπελάσουμε ένα κλειδί σ' ένα λεξικό, το οποίο δεν υπάρχει και θα προκαλέσει ένα KeyError. Ο τύπος της εξαίρεσης τυπώνεται.

Έξοδος: <class 'KeyError'>

31.4.6 Παράδειγμα 8

Εκτύπωση λεπτομερειών εξαίρεσης

Σ' αυτό το είδος σφάλματος χρησιμοποιούμε string formatting για να παρουσιάσουμε και το μήνυμα και την εξαίρεση. Αυτή η προσέγγιση παρέχει επίσης αναλυτική αναφορά του λάθους, τυπώνοντας τον τύπο του λάθους κι ένα διευκρινιστικό μήνυμα.

```
try:
    [] + "string"
except Exception as e:
    print(f"Error: {type(e).__name__}, Message: {str(e)}")
```

Στο παραπάνω παράδειγμα, προσπαθούμε να προσθέσουμε μια λίστα σε μια συμβολοσειρά κι έτσι έχουμε ένα TypeError. Τυπώνονται και το όνομα και το μήνυμα της εξαίρεσης.

Έξοδος: Error: TypeError, Message: can only concatenate list (not "str") to list

31.4.7 Παράδειγμα 9

Εκτύπωση γραμμής εξαίρεσης

Τυπώνοντας τον αριθμό της γραμμής ανακτούμε το ακριβές σημείο

```
import sys
try:
    x = y
except Exception as e:
    print(f"Error on line {sys.exc_info()[-1].tb_lineno}:
{str(e)}")
```

Στον παραπάνω κώδικα, αναφερόμαστε σε μια μεταβλητή η οποία δεν έχει αναφερθεί κι έτσι προκαλούμε ένα NameError. Τυπώνονται το λάθος και το μήνυμά του

Έξοδος: Error on line [line_number]: name 'y' is not defined

31.4.8 Παράδειγμα 10

Εκτύπωση Exception Message χωρίς Traceback

Εκτύπωση του μηνύματος λάθους χωρίς ανίχνευση και χρήση της print() για να δείξουμε το μήνυμα. Παρέχουμε έτσι ένα καθαρό και ακριβές μήνυμα χωρίς περαιτέρω πληροφορίες.

```
try:
    x = {}
    x.append(1)
except Exception as e:
    print(e)
```

Προσπαθούμε να χρησιμοποιήσουμε τη μέθοδο append (η οποία δεν εφαρμόζεται σε λεξικά) σε ένα λεξικό κι έτσι προκαλούμε ένα AttributeError. Το μήνυμα λάθους τυπώνεται χωρίς ανίχνευση.

Έξοδος: 'dict' object has no attribute 'append'

31.4.9 Παράδειγμα 11

Εκτύπωση της εξαίρεσης και συνέχιση της εκτέλεσης

Εκτύπωση της εξαίρεσης και συνέχιση της εκτέλεσης. Ανιχνεύουμε και τυπώνουμε την εξαίρεση ώστε να γνωρίζουμε πως υπάρχει ένα λάθος προς διόρθωση.

```
try:
    x = 1 / 0
except Exception as e:
    print(e)
print("Program continues...")
```

Σ' αυτό το παράδειγμα προσπαθούμε να διαιρέσουμε με το μηδέν κι έτσι προκαλούμε ένα ZeroDivisionError. Το μήνυμα τυπώνεται και το πρόγραμμα συνεχίζει.

Έξοδος: division by zero

Program continues...

31.4.9 Παράδειγμα 12

Εκτύπωση της εξαίρεσης σε μια γραμμή

Αυτό προϋποθέτει να χρησιμοποιήσουμε string formatting μέσα στη συνάρτηση print() για να δώσουμε ένα σαφές μήνυμα του λάθους.

```
try:
    x = [1, 2, 3]
    x[5]
except Exception as e:
    print(f"An error occurred: {e}")
```

Προσπαθούμε να προσπελάσουμε ένα index μιας λίστας το οποίο δεν υπάρχει κι έτσι παίρνουμε ένα IndexError το οποίο τυπώνεται σε μια γραμμή

Έξοδος: An error occurred: list index out of range

31.5.0 Διαχείριση αρχείων

Τα αρχεία κειμένου τα μεταχειριζόμαστε σαν τετράδια. Ένα τετράδιο πρώτα το ανοίγουμε και, όταν τελειώσουμε το γράψιμο, το κλείνουμε. Όταν είναι ανοικτό, μπορούμε να διαβάσουμε από αυτό ή να γράψουμε σε αυτό. Σε κάθε περίπτωση ξέρουμε πού ακριβώς βρισκόμαστε μέσα στο τετράδιο.

Όλα αυτά εφαρμόζονται και στα αρχεία. Για να ανοίξουμε ένα αρχείο, δηλώνουμε εάν θέλουμε να το διαβάσουμε (read) ή να το γράψουμε (write). Όταν τελειώσουμε την εργασία μας μ' αυτό, θα πρέπει να το κλείσουμε (close).

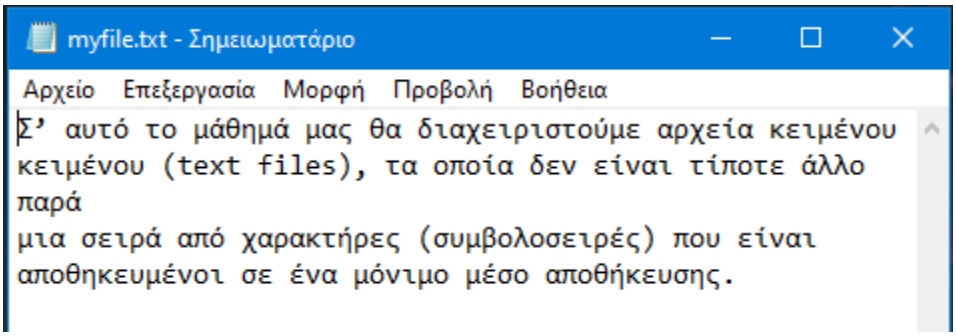
Ο πρώτος τύπος αρχείων τον οποίο θα διαβάσουμε, όπως είπαμε είναι τα απλά αρχεία κειμένου (.txt). Πρώτα ας δημιουργήσουμε ένα αρχειάκι

myfile.txt με λίγες γραμμές κειμένου (που δανειζόμαστε από το παραπάνω κείμενό μας):

«Σ’ αυτό το μάθημά μας θα διαχειριστούμε αρχεία κειμένου (text files), τα οποία δεν είναι τίποτε άλλο παρά μια σειρά από χαρακτήρες (συμβολοσειρές) που είναι αποθηκευμένοι σε ένα μόνιμο μέσο αποθήκευσης.»

```
f = open("C:\\Users\\NK\\Desktop\\myfile.txt", "w")
line1 = "Σ’ αυτό το μάθημά μας θα διαχειριστούμε αρχεία
κειμένου\n"
f.write(line1)
line2 = "κειμένου (text files), τα οποία δεν είναι τίποτε
άλλο παρά \n"
f.write(line2)
line3 = "μια σειρά από χαρακτήρες (συμβολοσειρές) που είναι
\n"
f.write(line3)
line4 = "αποθηκευμένοι σε ένα μόνιμο μέσο αποθήκευσης.\n"
f.write(line4)
f.close()
```

Έτσι δημιουργούμε το παρακάτω αρχείο:



Η συνάρτηση open δέχεται δύο ορίσματα: το πρώτο είναι το όνομα του αρχείου (χωρίς αναφορά μονοπατιού, εξ ορισμού ο φάκελος του αρχείου θα ήταν ο φάκελος στον οποίο είναι εγκατεστημένη η Python) και το δεύτερο όρισμα είναι ο τρόπος ανοίγματος (mode).

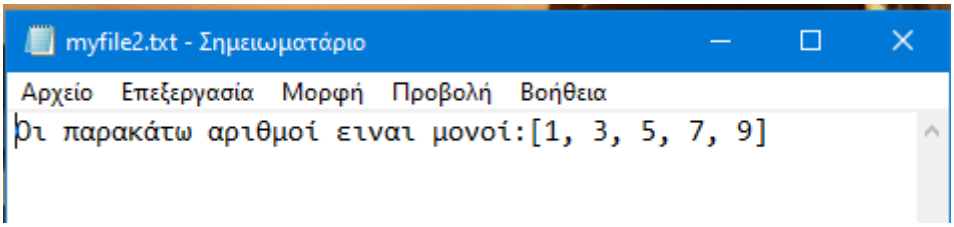
Το mode ‘w’ (write, εγγραφή) σημαίνει ότι ανοίγουμε το αρχείο για γράψιμο. Αν το αρχείο που ανοίγουμε για εγγραφή υπάρχει ήδη, τότε τα δεδομένα που περιέχει σβήνονται και αντικαθίστανται από τα νέα.
Αν το αρχείο δεν υπάρχει, τότε αυτό απλά δημιουργείται και είναι έτοιμο για να γράψουμε δεδομένα σε αυτό. Η εκτέλεση της εντολής open δημιουργεί ένα αντικείμενο αρχείου πάνω στο οποίο μπορούμε να καλέσουμε μεθόδους. Με τη μέθοδο write μπορούμε να γράψουμε δεδομένα σε ένα αρχείο.

Επίσης, το όρισμα της write πρέπει να είναι συμβολοσειρά. Έτσι, αν θέλουμε να γράψουμε και άλλες τιμές (π.χ. αριθμούς), θα πρέπει πρώτα να τις μετατρέψουμε σε συμβολοσειρές με χρήση της συνάρτησης str (casting).

Αν τρέξουμε κάποιες γραμμές κώδικα στον IDLE και καλέσουμε την write(), θα δούμε πως εμφανίζεται ο αριθμός των χαρακτήρων που γράφεται στο αρχείο με κάθε εντολή.
Παράδειγμα:

```
>>> f = open('C:\\Users\\NK\\Desktop\\myfile2.txt', 'w')
...
>>> f.write('Οι παρακάτω αριθμοί ειναι μονοί:')
...
32
>>> f.write(str([1,3,5,7,9]))
...
15
>>> f.close()
```

ενώ τώρα, το νέο αρχείο μας θα είναι:



31.5.1 Ανάγνωση αρχείων

Για να ανοίξουμε ένα αποθηκευμένο αρχείο κειμένου για ανάγνωση, πρέπει να χρησιμοποιήσουμε τον mode ‘r’ (read, ανάγνωση). Αν προσπαθήσουμε να ανοίξουμε για ανάγνωση ένα αρχείο που δεν υπάρχει, τότε θα εμφανιστεί μήνυμα λάθους. Το περιεχόμενο ενός αρχείου διαβάζεται ως συμβολοσειρά.

Τώρα που έχουμε μάθει να γράφουμε ένα αρχείο, πάμε να το ανοίξουμε για ανάγνωση.

```
f = open("C:\\Users\\NK\\Desktop\\myfile.txt", "r")
firstline = f.readline()
secondline = f.readline()
print (firstline)
print (secondline)
f.close()
```

Τα συχνά χρησιμοποιούμενα modes είναι:
'r' mode:
Μόνο για ανάγνωση.

'w' mode:
Μόνο για εγγραφή.
Αν το συγκεκριμένο αρχείο δεν υπάρχει, θα δημιουργηθεί.

Αν το συγκεκριμένο αρχείο υπάρχει, οτιδήποτε υπάρχει μέσα θα διαγραφεί.

'a' mode:
Για προσθήκη κειμένου.
Αν το συγκεκριμένο αρχείο δεν υπάρχει, θα δημιουργηθεί.
Αν υπάρχει, οτιδήποτε υπάρχει μέσα θα προστεθεί στο τέλος του αρχείου.

'r+' mode:
Και για γραφή και για ανάγνωση.

'x' mode:
Δημιουργεί ένα αρχείο κι επιστρέφει λάθος αν ήδη υπάρχει.

Αφού ανοίξουμε το αρχείο, η επόμενη δήλωση

firstline = f.readline()
διαβάζει την πρώτη γραμμή του αρχείου και την αναθέτει στη μεταβλητή firstline.
Κάθε φορά που καλείται η συνάρτηση readline(), διαβάζει μια νέα γραμμή από το αρχείο.
Στο παραπάνω προγραμματάκι η readline() καλείται τέσσερις φορές.
Έτσι, θα έχουμε την έξοδο:
Σ’ αυτό το μάθημά μας θα διαχειριστούμε αρχεία

κειμένου (text files), τα οποία δεν είναι τίποτε άλλο παρά

μια σειρά από χαρακτήρες (συμβολοσειρές) που είναι

αποθηκευμένοι σε ένα μόνιμο μέσο αποθήκευσης.

Θα παρατηρήσετε ότι μετά από κάθε γραμμή εισάγεται μια αλλαγή γραμμής. Αυτό συμβαίνει επειδή η συνάρτηση readline() προσθέτει τους χαρακτήρες '\n' στο τέλος κάθε γραμμής.
Αν δεν θέλουμε την επιπλέον γραμμή μεταξύ κάθε γραμμής κειμένου, μπορούμε να πούμε:
print (firstline, end= ''). Αυτό θα αφαιρέσει τους χαρακτήρες '\n'.
Σημείωση:

Το '' αποτελείται από δύο μονά εισαγωγικά κι όχι από ένα διπλό εισαγωγικό.

Έτσι τώρα έχουμε ένα πολύ πιο σωστό αποτέλεσμα εκτύπωσης:

Σ’ αυτό το μάθημά μας θα διαχειριστούμε αρχεία
κειμένου (text files), τα οποία δεν είναι τίποτε άλλο παρά
μια σειρά από χαρακτήρες (συμβολοσειρές) που είναι
αποθηκευμένοι σε ένα μόνιμο μέσο αποθήκευσης.

Αφού διαβάσουμε και εκτυπώσουμε τις τέσσερις πρώτες γραμμές, η τελευταία πρόταση ***f.close()*** κλείνει το αρχείο.

Πρέπει πάντα να κλείνουμε το αρχείο μόλις ολοκληρώσουμε την ανάγνωσή του για να ελευθερώνουμε τους πόρους του συστήματος.

Μπορούμε να δούμε και τα παρακάτω συμπληρωματικά παραδείγματα:

```
# Άνοιγμα και προσθήκη κειμένου στο αρχείο myfile.txt:
f = open("myfile.txt", "a")
f.write("Τώρα το αρχείο έχει περισσότερο περιεχόμενο!")
f.close()

# Άνοιγμα και διάβασμα του αρχείου μετά την προσθήκη
κειμένου:
f = open("myfile.txt", "r")
print(f.read())

# Άνοιγμα του αρχείου και επανωγράψιμο (το υπάρχον κείμενο
διαγράφεται):
f = open("myfile.txt", "w")
f.write("Ώπα! Έχουμε διαγράψει το περιεχόμενο!")
f.close()
```

Δημιουργία αρχείου:

Δημιουργία του αρχείου "myfile.txt":

f = open("myfile.txt", "x")

Το αποτέλεσμα είναι να δημιουργηθεί ένα κενό αρχείο!

Δημιουργία ενός αρχείου που δεν υπάρχει:

f = open("myfile.txt", "w")

31.6.0 Ασκήσεις

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.

Άσκηση 2: (factorial_8.11)

Το factorial ενός αριθμού, είναι ο αριθμός των γινομένων όλων των προηγούμενών 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.

Άσκηση 3: Προσάρτηση σε ένα αρχείο

Ανοίξτε το αρχείο "my_file.txt" και προσθέστε το παρακάτω κείμενο στο τέλος του αρχείου: "Αυτό είναι κείμενο που προστέθηκε στο αρχείο."

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.

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: "Αυτές είναι κάποιες σημειώσεις μου για την 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: "Έχουμε κάποιες επιπρόσθετες σημειώσεις για την 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: "Αυτό είναι το περιεχόμενο του πηγαίου αρχείου."

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

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