ΠΛΗΡΟΦΟΡΙΚΗ Ι (Python)
Ενότητα 6

Αρχεία (Files)

Η Python δίνει τη δυνατότητα εγγραφής δεδομένων σε αρχεία, καθώς και ανάγνωσης δεδομένων από αυτά. Ουσιαστικά πρόκειται για την είσοδο δεδομένων σε κάποιο πρόγραμμα, τα οποία είναι αποθηκευμένα σε κάποιο αρχείο, και για την έξοδο δεδομένων (αποτελεσμάτων) του προγράμματος μέσω της αποθήκευσής τους σε κάποιο αρχείο.

Σε αντίθεση με την είσοδο δεδομένων από το πληκτρολόγιο ή την έξοδο δεδομένων σε κάποια δομή δεδομένων (π.χ., μεταβλητή, λίστα κτλ), η είσοδος/έξοδος με τη χρήση αρχείων έχει τα εξής πλεονεκτήματα:

- Τα δεδομένα διατηρούνται και μετά το πέρας εκτέλεσης του προγράμματος (ή της λειτουργίας του κελύφους μέσω του οποίου καλείται το πρόγραμμα).
- Είναι πιο κατάλληλος τρόπος για την αποθήκευση μεγάλου όγκου δεδομένων.
- Είναι πιο κατάλληλος τρόπος για την επεξεργασία δεδομένων που έχουν προκύψει από άλλη εφαρμογή ή πρόγραμμα.

Το προς διαχείριση αρχείο “συνδέεται” με το πρόγραμμα μέσω ενός αντικειμένου αρχείου (file object). Το αντικείμενο που δημιουργείται στη μνήμη του υπολογιστή σχετίζεται με το συγκεκριμένο αρχείο και παρέχει έναν τρόπο για να δουλέψει το πρόγραμμα με αυτό, μέσω μιας μεταβλητής που αναφέρεται στο συγκεκριμένο αντικείμενο. Η μεταβλητή χρησιμοποιείται για τη διενέργεια των πράξεων που εκτελούνται στο αρχείο. Οι πράξεις αυτές έχουν να κάνουν με το άνοιγμα ενός αρχείου, την ανάγνωση δεδομένων από αυτό, την εγγραφή δεδομένων σε αυτό, και το κλείσιμό του. Πιο αναλυτικά:

Άνοιγμα αρχείου:

Για το άνοιγμα ενός αρχείου χρησιμοποιείται η εντολή `open`, ως εξής:

```
μεταβλητή_αρχείου = open(όνομα_αρχείου, προσπέλαση)
```

όπου η *`μεταβλητή_αρχείου`* είναι το όνομα της μεταβλητής που θα αναφέρεται στο αντικείμενο αρχείου, το *`όνομα_αρχείου`* είναι μια συμβολοσειρά που καθορίζει το όνομα του αρχείου και *`προσπέλαση`* είναι μια συμβολοσειρά που καθορίζει τον τρόπο προσπέλασης του αρχείου, δηλαδή εάν το αρχείο ανοίγεται για ανάγνωση δεδομένων από αυτό ή για εγγραφή δεδομένων σε αυτό (υπάρχουν και άλλες περιπτώσεις προσπέλασης, οι οποίες δεν καλύπτονται εδώ). Πιο συγκεκριμένα, οι αντίστοιχες τιμές του τελεστή προσπέλασης είναι οι εξής:

- `'r'` → *read-only:* Άνοιγμα αρχείου μόνο για ανάγνωση
- `'w'` → *write:* Άνοιγμα αρχείου για εγγραφή – Αν το αρχείο υπάρχει ήδη, τα περιεχόμενά του διαγράφονται. Αν δεν υπάρχει, δημιουργείται.
- `'a'` → *append-write:* Άνοιγμα αρχείου για εγγραφή – Αν το αρχείο υπάρχει ήδη, τα νέα δεδομένα εγγράφονται στο τέλος του. Αν δεν υπάρχει, δημιουργείται.

Σημείωση: Ιδιαίτερη προσοχή χρειάζεται στη χρήση του τελεστή `'w'`, διότι εάν το προς εγγραφή αρχείο υπάρχει ήδη αποθηκευμένο στον δίσκο, το υπάρχον περιεχόμενό του διαγράφεται.

Π.χ.,

α) Άνοιγμα αρχείου θερμοκρασιών με όνομα `temp.txt` για ανάγνωση των τιμών των θερμοκρασιών από ένα πρόγραμμα:

```
temperatures = open('temp.txt', 'r')
```

Μετά το άνοιγμα του αρχείου, η μεταβλητή `temperatures` αναφέρεται σε ένα αντικείμενο αρχείου το οποίο μπορεί να χρησιμοποιηθεί για την ανάγνωση των τιμών του αρχείου `temp.txt`.

β) Δημιουργία αρχείου `prog_output.txt` για την αποθήκευση των αποτελεσμάτων ενός προγράμματος. Σε περίπτωση που το αρχείο υπάρχει ήδη στον δίσκο, επιθυμούμε την αντικατάστασή του από το νέο αρχείο που θα δημιουργηθεί:

```
output = open('prog_output.txt', 'w')
```

Η εντολή δημιουργεί το αρχείο `prog_output.txt` και το ανοίγει. Μετά το άνοιγμα του αρχείου, η μεταβλητή `output` αναφέρεται σε ένα αντικείμενο αρχείου το οποίο μπορεί να χρησιμοποιηθεί για την εγγραφή τιμών στο αρχείο `prog_output.txt`. Σε περίπτωση που το αρχείο υπήρχε ήδη στον δίσκο, το περιεχόμενό του διαγράφεται πριν αρχίσουν οι εγγραφές.

γ) Αποθήκευση των αποτελεσμάτων ενός προγράμματος στο αρχείο `prog_output.txt`. Σε περίπτωση που το αρχείο υπάρχει ήδη στον δίσκο, τα νέα αποτελέσματα θα προστεθούν στο τέλος του, διαφορετικά το αρχείο θα δημιουργηθεί:

```
output = open('prog_output.txt', 'a')
```

Η εντολή ανοίγει το αρχείο `prog_output.txt` ή το δημιουργεί (εάν δεν υπάρχει ήδη) και το ανοίγει. Μετά το άνοιγμα του αρχείου, η μεταβλητή `output` αναφέρεται σε ένα αντικείμενο αρχείου το οποίο μπορεί να χρησιμοποιηθεί για την εγγραφή τιμών στο αρχείο `prog_output.txt`.

Καθορισμός της θέσης αρχείου στον δίσκο:

Για το άνοιγμα ενός αρχείου το οποίο δε βρίσκεται στον ίδιο υποκατάλογο με αυτόν που βρίσκεται το πρόγραμμα, θα πρέπει να καθοριστεί η ακριβής θέση του στον δίσκο, προσδιορίζοντας τη διαδρομή του (path) στο σύστημα αρχείων του λειτουργικού συστήματος. Π.χ., ένα αρχείο `test.txt` μπορεί να βρίσκεται στο ακόλουθο path (σε ένα σύστημα με λειτουργικό σύστημα Windows):

```
C:\Users\Blake\Data
```

Εάν το πρόγραμμα που ανοίγει το αρχείο `test.txt` δε βρίσκεται σε αυτόν τον υποκατάλογο, θα πρέπει να περιληφθεί ολόκληρο το path, π.χ. ως εξής:

```
test_file = open(r'C:\Users\Blake\Data\test.txt', 'w')
```

Σημείωση: Το `r` πριν από τη συμβολοσειρά υποδηλώνει ότι η συμβολοσειρά είναι μη-επεξεργάσιμη, που σημαίνει ότι ο διερμηνέας της Python ερμηνεύει τον χαρακτήρα τής ανάστροφης καθέτου (`\`) ως κανονικό χαρακτήρα τής συμβολοσειράς και όχι ως κάποια ακολουθία

διαφυγής (escape character).

Στην περίπτωση ενός συστήματος με λειτουργικό σύστημα Linux, ο προσδιορισμός του path δεν απαιτεί τη χρήση του χαρακτήρα `r`, αφού χρησιμοποιείται η κανονική κάθετος. Π.χ.:

```
test_file = open('/home/Blake/Data/test.txt', 'w')
```

Κλείσιμο αρχείου:

Όταν ολοκληρωθεί η διαχείριση ενός αρχείου (ανάγνωση ή εγγραφή) από ένα πρόγραμμα, θα πρέπει αυτό το αρχείο να κλείσει, ώστε να αποσυνδεθεί από το πρόγραμμα. Η παράλειψη του κλεισίματος ενός αρχείου μπορεί να οδηγήσει σε απώλεια δεδομένων. Για το κλείσιμο ενός αρχείου χρησιμοποιείται η μέθοδος `close`, η οποία εφαρμόζεται στη μεταβλητή που αναφέρεται στο αντικείμενο του προς κλείσιμο αρχείου:

```
μεταβλητή_αρχείου.close()
```

όπου η *μεταβλητή_αρχείου* είναι το όνομα της μεταβλητής που θα αναφέρεται στο αντικείμενο του συγκεκριμένου αρχείου.

Εγγραφή δεδομένων σε αρχείο:

Μετά το άνοιγμα ενός αρχείου, για την εγγραφή δεδομένων σε αυτό χρησιμοποιείται η μέθοδος `write`, ως εξής:

```
μεταβλητή_αρχείου.write(συμβολοσειρά)
```

όπου η *μεταβλητή_αρχείου* είναι το όνομα της μεταβλητής που αναφέρεται στο αντικείμενο του συγκεκριμένου αρχείου και *συμβολοσειρά* είναι η συμβολοσειρά που θα εγγραφεί στο αρχείο. Εάν το αρχείο δεν έχει ανοιχτεί με τον σωστό τρόπο προσπέλασης (`'w'` ή `'a'`), τότε θα προκληθεί σφάλμα.

Π.χ., εγγραφή τριών ονομάτων στο αρχείο `philosophers.txt`:

```
# Άνοιγμα αρχείου με όνομα philosophers.txt
outfile = open('philosophers.txt', 'w')

# Εγγραφή των ονομάτων τριών φιλοσόφων στο αρχείο
outfile.write('John Locke\n')
outfile.write('David Hume\n')
outfile.write('Edmund Burke\n')

# Κλείσιμο του αρχείου
outfile.close()
```

Ο χαρακτήρας `\n` στο τέλος της κάθε εγγραφής, κάνει την επόμενη εγγραφή να πραγματοποιηθεί σε νέα γραμμή στο αρχείο.

Ανάγνωση δεδομένων από αρχείο:

Μετά το άνοιγμα ενός αρχείου, για την ανάγνωση δεδομένων από αυτό μπορούν να χρησιμοποιηθούν οι μέθοδοι `read` και `readline`. Τα δεδομένα ενός αρχείου που διαβάζονται από κάποιο πρόγραμμα, αποθηκεύονται σε δομές δεδομένων (π.χ. μεταβλητές) του προγράμματος για να μπορούν στη συνέχεια να χρησιμοποιηθούν από το πρόγραμμα. Η μέθοδος `read` διαβάζει ολόκληρο το περιεχόμενο του αρχείου και το επιστρέφει ως μία συμβολοσειρά, ενώ η μέθοδος `readline` διαβάζει μία γραμμή (δηλαδή μέχρι να βρει τον χαρακτήρα νέας γραμμής `\n`). Με επαναλαμβανόμενες κλήσεις της μεθόδου `readline` μπορεί να διαβαστεί ολόκληρο το περιεχόμενο του αρχείου ανά γραμμή, αφού ο διερμηνέας χρησιμοποιεί μια ειδική τιμή γνωστή ως *θέση ανάγνωσης* για να γνωρίζει το σημείο μέχρι το οποίο έχει διαβαστεί κάποιο αρχείο (όταν ανοίγεται ένα αρχείο, η θέση ανάγνωσης βρίσκεται στην αρχή του αρχείου). Οι μέθοδοι ανάγνωσης χρησιμοποιούνται ως εξής:

```
μεταβλητή = μεταβλητή_αρχείου.read()
```

και

```
μεταβλητή = μεταβλητή_αρχείου.readline()
```

όπου η *μεταβλητή_αρχείου* είναι το όνομα της μεταβλητής που αναφέρεται στο αντικείμενο του συγκεκριμένου αρχείου και *μεταβλητή* είναι η μεταβλητή στην οποία θα αποθηκευτεί η συμβολοσειρά που επιστρέφουν οι μέθοδοι `read` ή `readline`.

Π.χ., για το αρχείο `philosophers.txt` του προηγούμενου παραδείγματος:

```
# Άνοιγμα αρχείου με όνομα philosophers.txt
infile = open('philosophers.txt', 'r')

# Ανάγνωση των περιεχομένων του αρχείου
file_contents = infile.read()

# Κλείσιμο του αρχείου
infile.close()

# Εμφάνιση των δεδομένων που διαβάστηκαν
print(file_contents)
```

Έξοδος του προγράμματος:

```
John Locke
David Hume
Edmund Burke
```

Αντίστοιχα, το αρχείο θα μπορούσε να διαβαστεί γραμμή-γραμμή, ως εξής:

```
# Άνοιγμα αρχείου με όνομα philosophers.txt
infile = open('philosophers.txt', 'r')

# Ανάγνωση τριών γραμμών από το αρχείο
line1 = infile.readline()
line2 = infile.readline()
```

```
line3 = infile.readline()

# Κλείσιμο του αρχείου
infile.close()

# Εμφάνιση των δεδομένων που διαβάστηκαν
print(line1)
print(line2)
print(line3)
```

Έξοδος του προγράμματος:

```
John Locke

David Hume

Edmund Burke
```

(Οι ενδιάμεσες γραμμές στην έξοδο του προγράμματος οφείλονται στο ότι η κάθε συμβολοσειρά περιέχει στο τέλος και τον χαρακτήρα νέας γραμμής `\n`*, επομένως σε κάθε* `print` *δημιουργείται μία νέα γραμμή από τον χαρακτήρα* `\n` *και μία από την ίδια την* `print`*).*

Εγγραφή συμβολοσειρών με προσάρτηση χαρακτήρα νέας γραμμής:

Όταν εγγράφονται πολλαπλά δεδομένα σε ένα αρχείο, για να εγγραφούν σε ξεχωριστές γραμμές ώστε να μπορούν να αναγνωστούν στη συνέχεια μεμονωμένα (με τη χρήση της `readline`), θα πρέπει να προσαρτηθεί στο τέλος τους ο χαρακτήρας νέας γραμμής (`\n`). Π.χ.,

```
# Εισαγωγή στοιχείων από το πληκτρολόγιο
name = input('Ονοματεπώνυμο: ')
father_name = input('Πατρώνυμο: ')
place_of_birth = input('Τόπος γέννησης: ')

# Άνοιγμα αρχείου με όνομα personal_data.txt
my_data = open('personal_data.txt', 'w')

# Εγγραφή των στοιχείων στο αρχείο (ανά γραμμή)
my_data.write(name + '\n')
my_data.write(father_name + '\n')
my_data.write(place_of_birth + '\n')

# Κλείσιμο του αρχείου
my_data.close()
```

Ανάγνωση συμβολοσειρών με αφαίρεση του χαρακτήρα νέας γραμμής:

Ο χαρακτήρας νέας γραμμής (\n) που υπάρχει στα δεδομένα που είναι αποθηκευμένα σε κάποιο αρχείο συνήθως υπάρχει απλά για να διαχωρίζονται τα δεδομένα και να μπορούν να διαβαστούν τμηματικά με τη μέθοδο readline. Επομένως, συνήθως είναι επιθυμητή η αφαίρεση του χαρακτήρα αυτού από τις συμβολοσειρές που τελικά διαβάζονται από το πρόγραμμα που ανοίγει και διαβάζει το αρχείο. Αυτό γίνεται με τη μέθοδο rstrip, η οποία καλείται με κάποια συμβολοσειρά και δέχεται

ως όρισμα μια συμβολοσειρά την οποία αφαιρεί από το τέλος της συμβολοσειράς με την οποία καλείται, και επιστρέφει τη νέα συμβολοσειρά που προκύπτει:

```
νέα_συμβολοσειρά = συμβολοσειρά.rstrip(συμβολοσειρά_προς_αποκοπή)
```

Π.χ.,

```
>>> s1 = 'Hello'
>>> s2 = s1.rstrip('o')
>>> print(s2)
Hell
```

Επομένως, μπορεί να χρησιμοποιηθεί για την αφαίρεση της συμβολοσειράς `'\n'`, π.χ. ως εξής:

```
# Άνοιγμα αρχείου με όνομα philosophers.txt
infile = open('philosophers.txt', 'r')

# Ανάγνωση τριών γραμμών από το αρχείο
line1 = infile.readline()
line2 = infile.readline()
line3 = infile.readline()

# Αφαίρεση του \n από κάθε συμβολοσειρά
line1 = line1.rstrip('\n')
line2 = line2.rstrip('\n')
line3 = line3.rstrip('\n')

# Κλείσιμο του αρχείου
infile.close()

# Εμφάνιση των δεδομένων που διαβάστηκαν
print(line1)
print(line2)
print(line3)
```

Έξοδος του προγράμματος:

```
John Locke
David Hume
Edmund Burke
```

(Σε αντίθεση με το αντίστοιχο παράδειγμα που παρουσιάστηκε προηγουμένως, τώρα δεν υπάρχουν επιπλέον ενδιάμεσες κενές γραμμές ανάμεσα στα ονόματα).

Εγγραφή και Ανάγνωση αριθμητικών δεδομένων

Τα δεδομένα στα αρχεία κειμένου αποθηκεύονται ως συμβολοσειρές, ακόμα και αν πρόκειται για αριθμητικές τιμές. Επομένως, για να αποθηκευτεί κάποιος αριθμός σε αρχείο κειμένου, πρέπει πρώτα να μετατραπεί σε συμβολοσειρά, με χρήση της συνάρτησης `str`. Π.χ.:

```
outfile = open('numbers.txt', 'w')
```

```
num1 = int(input('Δώσε έναν αριθμό: '))
num2 = int(input('Δώσε άλλον έναν αριθμό: '))
num3 = num1 + num2

# Εγγραφή των αριθμών στο αρχείο.
outfile.write(str(num1) + '\n')
outfile.write(str(num2) + '\n')
outfile.write(str(num3) + '\n')

outfile.close()
```

Αντίστοιχα, για να διαβαστεί ένας αριθμός από αρχείο και να χρησιμοποιηθεί ως αριθμός στο πρόγραμμα, θα πρέπει να μετατραπεί σε αριθμητική τιμή με τις συναρτήσεις `int` ή `float` (όπως δηλαδή γίνεται στην είσοδο αριθμητικών τιμών με τη συνάρτηση `input`). Π.χ., αν το αρχείο `numbers.txt` περιέχει ακέραιους αριθμούς, η είσοδος ενός ακεραίου μπορεί να γίνει ως εξής:

```
infile = open('numbers.txt', 'r')
value = int(infile.readline())
infile.close()
```

Σημείωση: Οι συναρτήσεις `int` και `float` έχουν την ιδιότητα να αγνοούν χαρακτήρες νέας γραμμής που τυχόν υπάρχουν στο τέλος κάποιας συμβολοσειράς.

Ανάγνωση αρχείου με βρόχο `while` ή `for`

Για την ανάγνωση των περιεχομένων ενός αρχείου μπορούν να χρησιμοποιηθούν βρόχοι. Με τη χρήση του βρόχου `while`, μπορούν να γίνουν επαναλαμβανόμενες αναγνώσεις μέχρι το τέλος του αρχείου, το οποίο σηματοδοτείται από την επιστροφή κενής συμβολοσειράς από τη μέθοδο `readline`. Π.χ., για την ανάγνωση όλων των αριθμητικών τιμών (π.χ., μέσων ημερήσιων θερμοκρασιών κάποιας περιοχής) που περιέχει το αρχείο `temperatures.txt`:

```
temp_file = open('temperatures.txt', 'r')

# Ανάγνωση της πρώτης γραμμής από το αρχείο, αλλά χωρίς να μετατραπεί
# ακόμα σε αριθμό. Πρέπει πρώτα να ελεγχθεί αν είναι η κενή συμβολοσειρά.
line = temp_file.readline()

# Όσο η readline δεν έχει επιστρέψει την κενή συμβολοσειρά,
# η επεξεργασία συνεχίζεται
while line != '':
    # Μετατροπή της line σε float (τιμή θερμοκρασίας)
    temp = float(line)
    # Μορφοποίηση και εκτύπωση της temp
    print(format(temp, '.2f'))
    # Ανάγνωση της επόμενης γραμμής.
    line = temp_file.readline()

temp_file.close()
```

Με τη χρήση ενός βρόχου `for` μπορεί να πραγματοποιηθεί η ίδια διαδικασία με πιο απλό τρόπο, με τον εξής τρόπο:

```
for μεταβλητή in αντικείμενο_αρχείου:
    εντολή
    εντολή
    ...
```

Το προηγούμενο παράδειγμα υλοποιείται με βρόχο `for` ως εξής:

```
temp_file = open('temperatures.txt', 'r')

# Ανάγνωση όλων των γραμμών του αρχείου
for line in temp_file:
    # Μετατροπή της line σε float (τιμή θερμοκρασίας)
    temp = float(line)
    # Μορφοποίηση και εκτύπωση της temp
    print(format(temp, '.2f'))

temp_file.close()
```

Εξαιρέσεις (Exceptions)

Εξαιρέσεις ονομάζονται κάποια σφάλματα που συμβαίνουν κατά την εκτέλεση ενός προγράμματος και τα οποία μπορεί κάποιος να τα διαχειριστεί με συγκεκριμένο τρόπο ώστε να αποφευχθεί ο απότομος τερματισμός του προγράμματος που θα προκαλούνταν σε διαφορετική περίπτωση. Η διαδικασία αυτή ονομάζεται “χειρισμός εξαιρέσεων” (exceptions handling) και υλοποιείται με την εντολή `try/except`. Η γενική και πιο απλή μορφή της εντολής αυτής είναι η εξής:

```
try:
    εντολή
    εντολή
    ...
except Όνομα_εξαίρεσης_1:
    εντολή
    εντολή
    ...
except Όνομα_εξαίρεσης_2:
    εντολή
    εντολή
    ...
...
```

Η εντολή αποτελείται από:

1. Το `try`-μπλοκ, το οποίο είναι το τμήμα κώδικα που ακολουθεί τη λέξη-κλειδί `try`, και το οποίο περιέχει τις εντολές του προγράμματος που μπορεί να προκαλέσουν κάποια εξαίρεση.
2. Τους χειριστές (handlers) της εξαίρεσης, που είναι τα μπλοκ κώδικα που ακολουθούν το `try`-μπλοκ, και ορίζονται με τη λέξη-κλειδί `except` ακολουθούμενη (προαιρετικά) από το όνομα μιας εξαίρεσης. Τα μπλοκ αυτά περιέχουν τον κώδικα που θα εκτελεστεί εάν προκληθεί κάποια συγκεκριμένη εξαίρεση στο `try`-μπλοκ.

Δηλαδή, αν κάποια εντολή του `try`-μπλοκ προκαλέσει κάποια εξαίρεση που αντιστοιχεί π.χ. στο *Όνομα_εξαίρεσης_1*, τότε διακόπτεται η κανονική ροή του προγράμματος και εκτελείται ο κώδικας που βρίσκεται στον χειριστή `except` του συγκεκριμένου σφάλματος (`except` *Όνομα_εξαίρεσης_1*), ενώ στη συνέχεια η ροή του προγράμματος συνεχίζεται με τον κώδικα που βρίσκεται μετά από ολόκληρη την εντολή `try/except`. Με τον τρόπο αυτόν, επιτυγχάνεται η διαχείριση του σφάλματος (της εξαίρεσης) χωρίς να τερματιστεί απότομα το πρόγραμμα. Σε πολλές περιπτώσεις, είναι επιθυμητός ο τερματισμός του προγράμματος μετά από κάποια εξαίρεση, όμως ο χειρισμός της βοηθάει τον προγραμματιστεί να εμφανίσει τα κατάλληλα μηνύματα στον χρήστη ή/και να εκτελέσει κάποιες απαραίτητες διεργασίες πριν τον τερματισμό του προγράμματος.

Τα ονόματα των εξαιρέσεων είναι συγκεκριμένα ονόματα που υποδηλώνουν τον τύπο σφάλματος που προκαλείται σε κάποια εξαίρεση, π.χ. `ValueError`, `IOError`. Η χρήση συγκεκριμένου ονόματος σε έναν χειριστή είναι προαιρετική. Εάν χρησιμοποιηθεί χειριστής χωρίς συγκεκριμένο όνομα εξαίρεσης, τότε είναι ένα γενικός χειριστής για οποιονδήποτε τύπο εξαίρεσης. Όταν υπάρχουν πολλοί χειριστές εξαιρέσεων σε μια εντολή `try/except`, ο διερμηνέας εξετάζει τους χειριστές με τη σειρά και όταν βρει κάποιον που να ταιριάζει με την εξαίρεση που προκλήθηκε, τότε τον εκτελεί και “βγαίνει” από την εντολή `try/except`. Για αυτόν τον λόγο, όταν υπάρχουν πολλοί χειριστές εξαιρέσεων σε μια εντολή `try/except`, τοποθετούνται από τον πιο ειδικό στον πιο γενικό (όπου αυτό είναι δυνατό), και επομένως εάν υπάρχει γενικός χειριστής (χωρίς όνομα εξαίρεσης), θα πρέπει να μπει τελευταίος (ο διερμηνέας της Python δεν επιτρέπει την τοποθέτησή του σε άλλη θέση).

Ο χειρισμός εξαιρέσεων αφορά σφάλματα που δεν μπορούν να αποφευχθούν με κάποιον απλούστερο τρόπο, δηλαδή με κάποιον έλεγχο `if/else`, όπως π.χ. θα ήταν μια διαίρεση με το 0. Το παρακάτω παράδειγμα παρουσιάζει μια τέτοια περίπτωση, όπου γίνεται χειρισμός της εξαίρεσης `ValueError` που προκαλείται σε περίπτωση που π.χ. ο χρήστης αντί για αριθμό πληκτρολογήσει κείμενο κατά την είσοδο από το πληκτρολόγιο:

```
def main():
    try:
        # Είσοδος των ωρών εργασίας.
        hours = int(input('Πόσες ώρες δούλεψες; '))

        # Είσοδος της ωριαία αμοιβής.
        pay_rate = float(input('Δώσε το ωρομίσθιο: '))

        # Υπολογισμός της ακαθάριστης αμοιβής.
        gross_pay = hours * pay_rate

        # Εμφάνιση της ακαθάριστης αμοιβής.
        print('Ακαθάριστη αμοιβή: €', format(gross_pay, '.2f'), sep='')

    except ValueError:
        print('ΣΦΑΛΜΑ: Οι ώρες εργασίες και το ωρομίσθιο')
        print('πρέπει να είναι έγκυροι αριθμοί')

main()
```

Μία συχνή εξαίρεση κατά την επεξεργασία αρχείων είναι η αποτυχία της εισόδου από αρχείο ή της εξόδου σε αρχείο, όπως π.χ. το να μην υπάρχει το αρχείο το οποίο προσπαθεί να ανοίξει ένα

πρόγραμμα. Ο χειρισμός μιας τέτοιας εξαίρεσης φαίνεται στο παρακάτω παράδειγμα (μαζί με τον χειρισμό μιας εξαίρεσης του τύπου που παρουσιάστηκε στο προηγούμενο παράδειγμα, καθώς και της γενικής εξαίρεσης):

```
def main():
    # Αρχικοποίηση ενός αθροιστή και ενός μετρητή.
    sum_temp = 0.0
    values = 0

    try:
        # Άνοιγμα αρχείου temperatures.txt.
        infile = open('temperatures.txt', 'r')

        # Ανάγνωση των τιμών από το αρχείο και άθροισή τους.
        for line in infile:
            temp = float(line)
            sum_temp += temp
            values += 1

        # Κλείσιμο του αρχείου.
        infile.close()

        # Εμφάνιση της μέσης θερμοκρασίας.
        print(format(sum_temp/values, '.2f'))

    except IOError:
        print('Προέκυψε σφάλμα στην ανάγνωση του αρχείου')

    except ValueError:
        print('Βρέθηκαν μη-αριθμητικά δεδομένα.')

    except:
        print('Παρουσιάστηκε σφάλμα.')

main()
```

<u>Προκαθορισμένο μήνυμα σφάλματος</u>

Όταν προκαλείται μια εξαίρεση, ουσιαστικά δημιουργείται ένα αντικείμενο, γνωστό ως αντικείμενο εξαίρεσης. Το αντικείμενο αυτό συνήθως περιέχει ένα προκαθορισμένο μήνυμα σφάλματος σχετικό με την εξαίρεση, το οποίο μπορεί να εμφανισθεί με την κλήση της συνάρτησης `print` με όρισμα το όνομα του αντικειμένου. Για να μπορεί να γίνει αυτό, ο χειριστής της εξαίρεσης θα πρέπει να προσδιορίζει ρητά το όνομα του αντικειμένου της εξαίρεσης, ως εξής:

```
except Όνομα_εξαίρεσης as αντικείμενο_εξαίρεσης:
```

οπότε, στη συνέχεια ο χειριστής μπορεί να περιέχει μια εντολή της μορφής:

```
print(αντικείμενο_εξαίρεσης)
```

για την εμφάνιση του προκαθορισμένου μηνύματος σφάλματος της συγκεκριμένης εξαίρεσης.

Π.χ., το παράδειγμα για την εξαίρεση `ValueError` παραπάνω, θα μπορούσε να γραφεί ως εξής:

```
def main():
    try:
        # Είσοδος των ωρών εργασίας.
        hours = int(input('Πόσες ώρες δούλεψες; '))

        # Είσοδος της ωριαία αμοιβής.
        pay_rate = float(input('Δώσε το ωρομίσθιο: '))

        # Υπολογισμός της ακαθάριστης αμοιβής.
        gross_pay = hours * pay_rate

        # Εμφάνιση της ακαθάριστης αμοιβής.
        print('Ακαθάριστη αμοιβή: €', format(gross_pay, '.2f'), sep='')

    except ValueError as err:
        print(err)

main()
```

οπότε, στο παρακάτω παράδειγμα λανθασμένης εισόδου (η λέξη 'σαράντα' αντί για την τιμή 40), το πρόγραμμα εμφανίζει το εξής:

```
Πόσες ώρες δούλεψες; σαράντα
invalid literal for int() with base 10: 'σαράντα'
```

Τα μπλοκ `else` και `finally`

Η εντολή `try/except` μπορεί να έχει ένα προαιρετικό μπλοκ `else`, το οποίο γράφεται μετά από όλους τους χειριστές `except`, και το οποίο εκτελείται αμέσως μετά τις εντολές του `try`-μπλοκ, *μόνο στην περίπτωση που δεν προκληθεί κάποια εξαίρεση*. Η γενική μορφή μιας εντολής `try/except` που περιλαμβάνει το μπλοκ `else` είναι η εξής:

```
try:
    εντολή
    εντολή
    ...
except Όνομα_εξαίρεσης_1:
    εντολή
    εντολή
    ...
except Όνομα_εξαίρεσης_2:
    εντολή
    εντολή
    ...
...
else:
    εντολή
    εντολή
    ...
```

Ο συγκεκριμένος τρόπος υλοποίησης έχει το πλεονέκτημα ότι μπορεί κάποιες εντολές που δεν θέλει ο προγραμματιστής να τις ελέγχει ως προς την πρόκληση εξαιρέσεων, να τις τοποθετεί στο `else`-μπλοκ αντί για το `try`-μπλοκ. Με αυτόν τον τρόπο αποφεύγεται ο ανεπιθύμητος χειρισμός εξαιρέσεων σε συγκεκριμένα τμήματα κώδικα, δίνοντας τη δυνατότητα να παραμένει μέσα στο `try`-μπλοκ μόνο το τμήμα κώδικα για το οποίο επιθυμεί ο προγραμματιστής να χειρίζεται τυχόν εξαιρέσεις που μπορεί να προκαλέσει.

Επίσης, η εντολή `try/except` μπορεί να έχει ένα προαιρετικό μπλοκ `finally`, το οποίο γράφεται μετά από όλους τους χειριστές `except`, και το οποίο εκτελείται πάντα (είτε συμβεί είτε δε συμβεί κάποια εξαίρεση), μετά την εκτέλεση του `try`-μπλοκ και όποιων χειριστών εξαιρέσεων τυχόν εκτελεστούν. Η γενική μορφή μιας εντολής `try/except` που περιλαμβάνει το μπλοκ `finally` είναι η εξής:

```
try:
    εντολή
    εντολή
    ...
except Όνομα_εξαίρεσης_1:
    εντολή
    εντολή
    ...
except Όνομα_εξαίρεσης_2:
    εντολή
    εντολή
    ...
...
finally:
    εντολή
    ...
```

<u>Παράδειγμα εξαιρέσεων</u>

Έχουμε το παρακάτω αρχείο `BB2014-15` που περιέχει τα ονόματα κάποιων παικτών που αγωνίστηκαν στο πρωτάθλημα μπάσκετ 2014-15, το σύνολο των πόντων που πέτυχαν και τον αριθμό των παιχνιδιών στα οποία συμμετείχαν. Π.χ.

```
ΜΑΥΡΟΕΙΔΗΣ 363 26
ΔΙΑΜΑΝΤΙΔΗΣ 211 25
ΣΠΑΝΟΥΛΗΣ 291 23
ΒΕΖΕΝΚΟΦ 469 26
ΖΑΡΑΣ 331 25
```

Θέλουμε να γράψουμε ένα πρόγραμμα που να υπολογίζει τον πρώτο σκόρερ και τον μέσο όρο πόντων που αυτός πέτυχε ανά παιχνίδι. Θέλουμε επίσης, για λόγους σύγκρισης, να εμφανίζει στο τέλος και τον μ.ο. πόντων του Νίκου Γκάλη (33.46) που είναι (και μάλλον θα είναι) ο πρώτος σκόρερ όλων των εποχών.

Έστω η παρακάτω υλοποίηση που χειρίζεται κάποιες από τις εξαιρέσεις που μπορεί να εμφανιστούν κατά την εκτέλεσή του. Οι πιθανές εξαιρέσεις είναι 4 και σημειώνονται ως σχόλια στο πρόγραμμα στις θέσεις στις οποίες μπορεί να προκύψουν. Έχουμε επιλέξει να χειριστούμε τις 3 από αυτές.

```
file_name = input('Αρχείο παικτών στο πρωτάθλημα μπάσκετ 2014-15? ')
print('-'*60)

try:
    f = open(file_name, 'r')  # Πιθανή εξαίρεση IOError

    most_points = 0
    for line in f:

        # Εξαίρεση IndexError αν len(player_data)<3
        player_data = line.split()
        name = player_data[0]
        points_scored = int(player_data[1])  # Πιθανή εξαίρεση ValueError
        games_played = int(player_data[2])   # Πιθανή εξαίρεση ValueError

        if points_scored > most_points:
            first_scorer = name
            most_points = points_scored
            # Πιθανή διαίρεση με 0 (ZeroDivisionError)
            average_points_per_game = points_scored/games_played

except IOError:
    print('Σφάλμα στην ανάγνωση του αρχείου', file_name)

except ValueError as error:
    print(error)

except IndexError:
    print('Ελλιπής αριθμών πεδίων σε κάποια γραμμή του αρχείου')

else:
    f.close()
    print('Πρώτος σκόρερ του πρωταθλήματος:', first_scorer)
    print('Μ.Ο. πόντων ανά αγώνα:',format(average_points_per_game,'.2f'))

finally:
    print('-'*60)

print('Πρώτος σκόρερ όλων των εποχών: ΓΚΑΛΗΣ')
print('Μέσος όρος πόντων ανά παιχνίδι:', 33.46)
```

Αν εκτελέσουμε το πρόγραμμα με είσοδο το παραπάνω αρχείο `BB2014-15` εμφανίζεται:

```
------------------------------------------------------------
Πρώτος σκόρερ του πρωταθλήματος:  ΒΕΖΕΝΚΟΦ
Μ.Ο. πόντων ανά αγώνα: 18.04
------------------------------------------------------------
Πρώτος σκόρερ όλων των εποχών: ΓΚΑΛΗΣ
Μέσος όρος πόντων ανά παιχνίδι: 33.46
```

Αν δώσουμε στην είσοδο το όνομα ενός αρχείου παικτών το οποίο δεν μπορούμε να το επεξεργαστούμε (π.χ. αν δεν υπάρχει αρχείο με όνομα `FOO.TXT` που δίνεται παρακάτω) τότε προκύπτει μια εξαίρεση `IOError` και η έξοδος του προγράμματος είναι:

```
------------------------------------------------------------
Σφάλμα στην ανάγνωση του αρχείου FOO.TXT
------------------------------------------------------------
Πρώτος σκόρερ όλων των εποχών: ΓΚΑΛΗΣ
Μέσος όρος πόντων ανά παιχνίδι: 33.46
```

Αν δώσουμε ως είσοδο το αρχείο:

```
ΜΑΥΡΟΕΙΔΗΣ 363 26
ΔΙΑΜΑΝΤΙΔΗΣ 25
ΣΠΑΝΟΥΛΗΣ 291 23
ΒΕΖΕΝΚΟΦ 469 26
ΖΑΡΑΣ 331 25
```

στο οποίο η 2η γραμμή έχει μόνο 2 πεδία (το όνομα και τη συμμετοχή του παίκτη) αντί για 3, τότε προκύπτει μια εξαίρεση `IndexError` γιατί η λίστα `player_data` για τη γραμμή 2 έχει μήκος 2 και το πρόγραμμα χρειάζεται και το 3 στοιχείο της. Η έξοδος στην περίπτωση αυτή είναι:

```
------------------------------------------------------------
Ελλιπής αριθμών πεδίων σε κάποια γραμμή του αρχείου
------------------------------------------------------------
Πρώτος σκόρερ όλων των εποχών: ΓΚΑΛΗΣ
Μέσος όρος πόντων ανά παιχνίδι: 33.46
```

Αν δώσουμε ως είσοδο το αρχείο:

```
ΜΑΥΡΟΕΙΔΗΣ 363 26
ΔΙΑΜΑΝΤΙΔΗΣ 211 25
ΣΠΑΝΟΥΛΗΣ 291 23.14159
ΒΕΖΕΝΚΟΦ 469 26
ΖΑΡΑΣ 331 25
```

στο οποίο φαίνεται ότι ο Σπανούλης έχει αγωνιστεί σε 23.14159 παιχνίδια, θα προκύψει μια εξαίρεση `ValueError` την οποία χειριζόμαστε με την εμφάνιση του προκαθορισμένου μηνύματος σφάλματος της συγκεκριμένης εξαίρεσης, με έξοδο:

```
------------------------------------------------------------
invalid literal for int() with base 10: '23.14159'
------------------------------------------------------------
Πρώτος σκόρερ όλων των εποχών: ΓΚΑΛΗΣ
Μέσος όρος πόντων ανά παιχνίδι: 33.46
```

Τέλος, αν στο αρχείο εισόδου έχει καταχωρηθεί λανθασμένα ότι ο Βεζένκοφ έχει αγωνιστεί σε 0 παιχνίδια, αλλά έχει σημειώσει πόντους,

```
ΜΑΥΡΟΕΙΔΗΣ 363 26
ΔΙΑΜΑΝΤΙΔΗΣ 211 25
ΣΠΑΝΟΥΛΗΣ 291 23
ΒΕΖΕΝΚΟΦ 469 0
ΖΑΡΑΣ 331 25
```

τότε στο πρόγραμμα θα γίνει διαίρεση με το 0 και θα προκύψει μια εξαίρεση `ZeroDivisionError` την οποία έχουμε επιλέξει να μην χειριστούμε οπότε το πρόγραμμα

τερματίζει απότομα με την εμφάνιση του μηνύματος:

```
------------------------------------------------------------
------------------------------------------------------------
Traceback (most recent call last):
  File "cs1sec6.py", line 16, in <module>
    average_points_per_game = points_scored/games_played
ZeroDivisionError: division by zero
```

Παρατηρούμε ότι σε όλες τις περιπτώσεις *εκτός της τελευταίας*, εκτελείται τόσο η `finally` (που εμφανίζει τη 2η οριζόντια γραμμή από παύλες) όσο και οι γραμμές που ακολουθούν το `try`-μπλοκ, γιατί στις περιπτώσεις αυτές έχουμε επιλέξει να χειριστούμε τις εξαιρέσεις που προέκυψαν. Το μπλοκ `else` εκτελείται *μόνο* στην περίπτωση που στο πρόγραμμά μας δεν προκύψει κάποια εξαίρεση. Αξίζει να σημειωθεί ότι η `finally` εκτελείται ακόμα και στην περίπτωση που η εξαίρεση που προκύπτει δεν χειρίζεται από κάποιο από τα υπάρχοντα `except`, όπως φαίνεται στην έξοδο του τελευταίου παραδείγματος εισόδου, όπου πρώτα εκτελείται η `finally` και στη συνέχεια εμφανίζεται το μήνυμα σφάλματος και τερματίζεται το πρόγραμμα.