



Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ

Τεχνικές Προγραμματισμού Υπολογιστών

Εργαστηριακό μάθημα • Α' εξάμηνο

Python

Σημειώσεις

Οδηγός μελέτης και εργαστηρίου

Από τις πρώτες εντολές στη σχεδίαση, την κατανόηση και τον έλεγχο ολοκληρωμένων προγραμμάτων.

ΜΑΘΑΙΝΩ • ΔΟΚΙΜΑΖΩ • ΚΑΤΑΝΟΩ, Απλές εξηγήσεις, έγχρωμα παραδείγματα, ελληνικά σχόλια και ασκήσεις με αιτιολογημένες απαντήσεις.

Καταλαβαίνω	Εφαρμόζω	Ελέγχω
Τιμή, μεταβλητή, συνθήκη, επανάληψη.	Συνάρτηση, συλλογή δεδομένων, αντικείμενο.	Παράδειγμα με το χέρι, οριακή δοκιμή, διόρθωση.

Python με είσοδο πληκτρολογίου, κονσόλα και αρχεία κειμένου.

Συναρτήσεις • Δεδομένα • Αντικείμενα
Αλγόριθμοι • Σχεδίαση • Δοκιμές

Διδάσκων: **Πεφάνης Ευάγγελος**

Για το εργαστήριο και για αυτοτελή μελέτη.

Πώς να χρησιμοποιήσετε τις σημειώσεις

Οι σημειώσεις απευθύνονται σε φοιτητές που ξεκινούν τώρα τον προγραμματισμό. Δεν χρειάζεται να γνωρίζετε ήδη όλα τα εργαλεία της γλώσσας. Σε κάθε κεφάλαιο μαθαίνετε μία ιδέα, παρακολουθείτε ένα μικρό πρόγραμμα και ελέγχετε αν μπορείτε να εξηγήσετε το αποτέλεσμα του με δικά σας λόγια.

Προτεινόμενη πορεία, Διαβάζω τον στόχο → προβλέπω την έξοδο → γράφω και εκτελώ τον κώδικα → αλλάζω ένα δεδομένο → ελέγχω και αιτιολογώ τη διαφορά.

Τα πλαίσια «Δοκιμάστε πριν διαβάσετε την απάντηση» συνοδεύονται από αιτιολογημένη λύση. Καλύψτε προσωρινά την απάντηση και προσπαθήστε πρώτα μόνοι σας. Οι αναφορές στο τέλος κάθε κεφαλαίου οδηγούν στις επίσημες πηγές στο τέλος του βιβλίου.

Τα κεφάλαια που σημειώνονται «Προαιρετική εμβάθυνση» δεν είναι προϋπόθεση για να ξεκινήσετε. Εξηγούν εργαλεία που γίνονται χρήσιμα όταν οργανώνετε μεγαλύτερα προγράμματα. Οι σημειώσεις καλύπτουν τον εκπαιδευτικό κορμό του μαθήματος· δεν είναι κατάλογος κάθε λειτουργίας του προτύπου της γλώσσας.

Υπάρχουν 42 αριθμημένα παραδείγματα. Το κάθε ένα συνοδεύεται από αρχείο στον φάκελο Examples/Python. Η ένδειξη εισόδου περιγράφει τι πληκτρολογούμε και το πλαίσιο εξόδου δείχνει τι παρήγαγε η αντίστοιχη δοκιμή. Οι επιπλέον ενδείξεις του περιβάλλοντος εκτέλεσης δεν αποτελούν μέρος της εξόδου.

Οι τίτλοι του πίνακα που ακολουθεί είναι σύνδεσμοι προς τα κεφάλαια. Μπορείτε επίσης να χρησιμοποιήσετε το παράθυρο περιήγησης του Word για τις επικεφαλίδες.

Χάρτης του εγχειριδίου

Κεφ.	Τι θα μελετήσετε	Πορεία
1	Το πρώτο πρόγραμμα και ο τρόπος εργασίας	Βασικός κορμός
2	Μεταβλητές, τιμές και βασικοί τύποι	Βασικός κορμός
3	Είσοδος, έξοδος και μορφοποίηση	Βασικός κορμός
4	Αριθμητικές πράξεις και μονάδες μέτρησης	Βασικός κορμός
5	Συνθήκες και λογική	Βασικός κορμός
6	for, range, while και παρακολούθηση βημάτων	Βασικός κορμός
7	Συναρτήσεις με καθαρές ευθύνες	Βασικός κορμός
8	Λίστες, θέσεις και πραγματικά αντίγραφα	Βασικός κορμός
9	Στατιστικά, φίλτρα και ταξινόμηση	Βασικός κορμός
10	Λεξικά, πλειάδες και σύνολα	Βασικός κορμός
11	Ανθεκτική είσοδος και εξαιρέσεις	Βασικός κορμός
12	Αρθρώματα, math, random και οργάνωση αρχείου	Βασικός κορμός

Κεφ.	Τι θα μελετήσετε	Πορεία
13	Αρχεία κειμένου, JSON και CSV	Βασικός κορμός
14	Κλάσεις, self και ελεγχόμενες ιδιότητες	Βασικός κορμός
15	Λίστες αντικειμένων και επιλογή κατάλληλης λύσης	Βασικός κορμός
16	Κατάσταση εφαρμογής και αμετάβλητοι κανόνες	Βασικός κορμός
17	Αναζήτηση κατάλληλης λύσης και προσανατολισμοί	Εμβάθυνση
18	Αναδρομή και αναζήτηση με επιστροφή	Εμβάθυνση
19	Δοκιμές, σφάλματα και τεκμηρίωση	Βασικός κορμός
20	Ολοκληρωμένο παράδειγμα: αναφορά βαθμολογίας	Βασικός κορμός
21	Χάρτης μελέτης για τα τέσσερα θέματα	Βασικός κορμός

01 | Το πρώτο πρόγραμμα και ο τρόπος εργασίας

Στόχος, Να γράψετε και να εκτελείτε ένα αρχείο .py και να ξεχωρίζετε τον κώδικα από τα αποτελέσματά του.

Ένα πρόγραμμα παίρνει δεδομένα, εφαρμόζει κανόνες και παράγει αποτέλεσμα. Για να υπολογίσουμε έναν μέσο χρόνο πρέπει να ξέρουμε ποιοι χρόνοι συμμετέχουν, πώς προστίθενται και με ποιο πλήθος θα διαιρεθούν. Αυτή η ακριβής περιγραφή βημάτων λέγεται αλγόριθμος. Η Python είναι η γλώσσα στην οποία θα εκφράσουμε τα βήματα.

Εκτέλεση με IDLE

1. Ανοίξτε το IDLE του εργαστηρίου. Το παράθυρο Shell επιτρέπει μικρές δοκιμές και εμφανίζει αποτελέσματα.
2. Επιλέξτε File → New File. Γράψτε το πρόγραμμα στο νέο παράθυρο επεξεργασίας, όχι τις ενδείξεις >>> του Shell.
3. Αποθηκεύστε με όνομα όπως first_program.py. Η επέκταση πρέπει να είναι .py, όχι .py.txt.
4. Επιλέξτε Run → Run Module ή F5. Οι εκτυπώσεις εμφανίζονται στο Shell. Σε ερώτηση input πληκτρολογείτε απάντηση και Enter.
5. Αλλάξτε μια τιμή και εκτελέστε ξανά ολόκληρο το αρχείο. Ελέγξτε ότι λειτουργεί από καθαρή αρχική κατάσταση.

ΠΑΡΑΔΕΙΓΜΑ PY01, Μια μικρή αναφορά

```
# Οι χρόνοι είναι αριθμοί σε λεπτά και όχι κείμενα.
first = 12
second = 15
third = 9
average = (first + second + third) / 3
print("Μέσος χρόνος:", average, "λεπτά")
```

Έξοδος της συγκεκριμένης δοκιμής

Μέσος χρόνος: 12.0 λεπτά

Η `print` είναι η εντολή εμφάνισης. Η ανάθεση με `=` αποθηκεύει μια τιμή. Το αρχείο εκτελεί τις εντολές από πάνω προς τα κάτω, εκτός αν κάποια δομή αλλάξει τη ροή.

Εναλλακτικά μπορείτε να εκτελέσετε το αρχείο από τερματικό, μέσα στον φάκελό του, με `python first_program.py` ή, σε εγκατάσταση Windows που διαθέτει τον launcher, `py first_program.py`. Το όνομα της εντολής εξαρτάται από την εγκατάσταση. Τα παραδείγματα χρησιμοποιούν Python 3.10 ή νεότερη και βιβλιοθήκες που περιλαμβάνονται στην Python.

Κώδικας και έξοδος, Στα πλαίσια κώδικα δεν γράφουμε `>>>` ούτε αντιγράφουμε τις γραμμές αποτελεσμάτων μέσα στο πρόγραμμα. Τα αποτελέσματα έχουν δικό τους πλαίσιο. Οι εσοχές έχουν σημασία: χρησιμοποιούμε σταθερά τέσσερα κενά και δεν αναμειγνύουμε tabs με κενά.

Το αρχείο `.py` είναι απλό κείμενο. Τα χρώματα τα εμφανίζει ο επεξεργαστής ή οι σημειώσεις· δεν αποτελούν εντολές της γλώσσας. Οι συνηθισμένες λέξεις της Python γράφονται στα αγγλικά, αλλά τα μηνύματα και τα σχόλια μπορούν να είναι πλήρως ελληνικά.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί είναι χρήσιμο να δοκιμάζουμε ολόκληρο το αρχείο και όχι μόνο γραμμές στο Shell;

Απάντηση και αιτιολόγηση, Ένα αρχείο που εκτελείται μόνο επειδή υπάρχουν παλιές μεταβλητές στο Shell δεν είναι αυτοτελές. Η καθαρή εκτέλεση αποκαλύπτει ελλείψεις στην αρχικοποίηση και στη σειρά εντολών.

Τεκμηρίωση κεφαλαίου: [P1], [P2].

02 | Μεταβλητές, τιμές και βασικοί τύποι

Στόχος, Να ξεχωρίζετε αριθμούς, κείμενα και λογικές τιμές και να καταλαβαίνετε τη λειτουργία της ανάθεσης.

Μια μεταβλητή είναι ένα όνομα που συνδέεται με μια τιμή. Δεν χρειάζεται να δηλώσουμε εκ των προτέρων τον τύπο της. Ωστόσο, πρέπει εμείς να καταλαβαίνουμε τι αντιπροσωπεύει κάθε όνομα. Αν το `quantity` σημαίνει πλήθος τεμαχίων, δεν πρέπει αργότερα να του αναθέσουμε ένα ονοματεπώνυμο.

Τύπος	Παράδειγμα	Χρήση
<code>int</code>	<code>12, -3</code>	Ακέραια πλήθη και θέσεις.
<code>float</code>	<code>2.5, 0.0</code>	Μετρήσεις και δεκαδικοί υπολογισμοί.
<code>str</code>	<code>"0012", "Άννα"</code>	Κείμενα και κωδικοί.
<code>bool</code>	<code>True, False</code>	Απάντηση σε λογική ερώτηση.
<code>None</code>	<code>selected = None</code>	Δήλωση ότι δεν υπάρχει επιλογή ή αποτέλεσμα.

ΠΑΡΑΔΕΙΓΜΑ PY02, Η νέα τιμή προκύπτει από την παλιά

```
# Η δεξιά πλευρά υπολογίζεται με την υπάρχουσα τιμή. Έπειτα το ίδιο όνομα
# συνδέεται με το νέο αποτέλεσμα.
water = 40
water = water - 7
water = water + 2
print(water)
print(type(water).__name__)
print(type("35").__name__)
```

Έξοδος της συγκεκριμένης δοκιμής

```
35
int
```

```
str
```

Πρώτα υπολογίζεται η δεξιά πλευρά και μετά συνδέεται το όνομα με τη νέα τιμή. Το "35" με εισαγωγικά είναι κείμενο, όχι αριθμός.

Η Python ξεχωρίζει κεφαλαία και πεζά: price και Price είναι διαφορετικά ονόματα. Επιλέγουμε σύντομα αλλά σαφή ονόματα όπως student_name, daily_energy και total. Αποφεύγουμε ονόματα όπως list, sum και input για δικές μας μεταβλητές, επειδή έτσι κρύβουμε έτοιμες λειτουργίες της γλώσσας.

ΠΑΡΑΔΕΙΓΜΑ PY03, Τι αντιγράφει η ανάθεση ενός αριθμού

```
a = 4
b = a
# Η νέα ανάθεση του a δεν αλλάζει την τιμή που έχει το b.
a = 9
print(a, b)
selected = None
print(selected is None)
```

Έξοδος της συγκεκριμένης δοκιμής

```
9 4
True
```

Το is None είναι σαφής έλεγχος απουσίας τιμής. Δεν το αντικαθιστούμε με if not selected όταν το 0 μπορεί να αποτελεί κανονικό αποτέλεσμα.

Πρώτη γνωριμία με αναφορές, Στις λίστες και στα λεξικά δύο ονόματα μπορεί να αναφέρονται στο ίδιο μεταβαλλόμενο αντικείμενο. Αυτό διαφέρει από το παραπάνω παράδειγμα νέας ανάθεσης αριθμού. Θα το εξετάσουμε με συγκεκριμένες αλλαγές, όχι μόνο με ορισμούς.

Δοκιμάστε πριν διαβάσετε την απάντηση, Πρέπει ο αριθμός μητρώου "0012" να μετατραπεί σε int;

Απάντηση και αιτιολόγηση, Όχι όταν χρησιμοποιείται ως αναγνωριστικό και πρέπει να διατηρηθούν τα αρχικά μηδενικά. Το "0012" και το "12" μπορεί να είναι διαφορετικοί κωδικοί σύμφωνα με την εκφώνηση.

Τεκμηρίωση κεφαλαίου: [P2], [P19].

03 | Είσοδος, έξοδος και μορφοποίηση

Στόχος, Να παίρνετε απαντήσεις με input, να μετατρέπετε τύπους και να εμφανίζετε σαφείς αναφορές.

Η input επιστρέφει πάντα κείμενο όταν η ανάγνωση ολοκληρώνεται. Η int μετατρέπει γραφή ακεραίου και η float γραφή δεκαδικού. Η μετατροπή είναι απαραίτητη πριν από πράξεις. Το print μπορεί να εμφανίσει πολλά ορίσματα με κενά ανάμεσά τους. Με f-string ενσωματώνουμε μεταβλητές μέσα σε ένα μορφοποιημένο κείμενο.

ΠΑΡΑΔΕΙΓΜΑ PY04, Μία έγκυρη αριθμητική είσοδος

```
# Στο παράδειγμα θεωρούμε ότι πληκτρολογείται αποδεκτός αριθμός.
raw = input("Μήκος σε μέτρα: ")
meters = float(raw.strip().replace(",", "."))
centimeters = meters * 100
print(f"Μήκος: {centimeters:.1f} cm")
```

Δοκιμή εισόδου, με τη σειρά: 1,5.

Έξοδος της συγκεκριμένης δοκιμής

```
Μήκος: 150.0 cm
```

Για είσοδο 1,5 η strip αφαιρεί εξωτερικά κενά και η replace μετατρέπει το κόμμα σε τελεία. Ο πλήρης έλεγχος άκυρης εισόδου θα προστεθεί στο κεφάλαιο 11.

Γραφή	Τι κάνει
<code>text.strip()</code>	Αφαιρεί εξωτερικά κενά και αλλαγές γραμμής.
<code>float(text)</code>	Μετατρέπει αριθμητικό κείμενο σε float.
<code>int(text)</code>	Μετατρέπει γραφή ακεραίου σε int.
<code>f"{value:.2f}"</code>	Εμφανίζει δύο δεκαδικά.
<code>f"{value:+.2f}"</code>	Εμφανίζει και πρόσημο, π.χ. +2.00.
<code>f"{name:<15}"</code>	Ευθυγραμμίζει κείμενο αριστερά σε πεδίο πλάτους 15.

ΠΑΡΑΔΕΙΓΜΑ PY05, Μια αναφορά με σαφείς στήλες

```
# Η μορφοποίηση ευθυγραμμίζει την αναφορά για ανάγνωση. Δεν αλλάζει την
# αποθηκευμένη αριθμητική τιμή.
name = "Αισθητήρας"
value = 23.456
change = -0.25
print(f"{'Όνομα':<15} {'Τιμή':>8} {'Μεταβολή':>10}")
print(f"{'name':<15} {'value':>8.2f} {'change':>+10.2f}")
print("Η αποθηκευμένη μέτρηση παραμένει:", value)
```

Έξοδος της συγκεκριμένης δοκιμής

```
Όνομα      Τιμή  Μεταβολή
Αισθητήρας 23.46  -0.25
Η αποθηκευμένη μέτρηση παραμένει: 23.456
```

Η μορφοποίηση αφορά την εμφάνιση. Δεν αντικαθιστά τη μεταβλητή `value` ούτε κάνει τις επόμενες συγκρίσεις πάνω στη στρογγυλοποιημένη γραφή.

Το `int("2.5")` αποτυγχάνει: το κείμενο δεν είναι γραφή ακεραίου. Το `int(2.5)`, αντίθετα, δίνει 2, επειδή δέχεται ήδη αριθμητική τιμή και αποκόπτει το κλασματικό μέρος προς το μηδέν. Δεν χρησιμοποιούμε αυτή τη δεύτερη συμπεριφορά για να αποδεχθούμε κρυφά μια λανθασμένη απάντηση σε ερώτηση πλήθους.

Μην εκτελείτε την είσοδο ως κώδικα, Η `eval` δεν είναι λύση για ανάγνωση αριθμού ή επιλογή πράξης. Η εφαρμογή πρέπει να μετατρέπει δεδομένα, όχι να εκτελεί αυθαίρετες εντολές που πληκτρολόγησε ο χρήστης.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί το `print("Σύνολο:", "2" + "3")` εμφανίζει 23 και όχι 5;

Απάντηση και αιτιολόγηση, Επειδή τα δύο ορίσματα της πρόσθεσης είναι κείμενα. Το + τα ενώνει. Χρειάζεται μετατροπή σε αριθμούς όταν ζητείται αριθμητικό άθροισμα.

Τεκμηρίωση κεφαλαίου: [P5], [P15].

04 | Αριθμητικές πράξεις και μονάδες μέτρησης

Στόχος, Να μεταφράζετε τύπους, να ξεχωρίζετε διαίρεση και υπόλοιπο και να χειρίζεστε δεκαδική ακρίβεια.

Ο σωστός υπολογισμός προϋποθέτει σωστό μοντέλο. Ισχύς σε W δεν είναι ενέργεια σε kWh. Μήκος σε cm δεν μπορεί να αντικατασταθεί αδιακρίτως σε τύπο που περιμένει μέτρα. Γράφετε μονάδες στα ονόματα ή στα σχόλια των παραμέτρων και στις τελικές εκτυπώσεις.

Τελεστής	Παράδειγμα	Αποτέλεσμα
/	7 / 2	3.5
//	7 // 2	3

Τελεστής	Παράδειγμα	Αποτέλεσμα
%	7 % 2	1
**	3 ** 2	9
- με παρενθέσεις	(-3) ** 2	9

ΠΑΡΑΔΕΙΓΜΑ ΡΥ06, Διάρθρωση προς τα κάτω και υπόλοιπο

```
# Η // στρογγυλοποιεί προς τα κάτω και όχι προς το μηδέν. Αυτό γίνεται
# φανερό όταν εξετάσουμε αρνητικό αριθμό.
items = 23
box_size = 5
print("Γεμάτα κουτιά:", items // box_size)
print("Υπόλοιπο:", items % box_size)
print(-7 // 2)
print(int(-7 / 2))
```

Έξοδος της συγκεκριμένης δοκιμής

```
Γεμάτα κουτιά: 4
Υπόλοιπο: 3
-4
-3
```

Η // στρογγυλοποιεί προς τα κάτω: $-7 // 2$ είναι -4 . Η $\text{int}(-3.5)$ δίνει -3 , προς το μηδέν. Για μη αρνητικά πλήθη η διαφορά δεν φαίνεται, αλλά για αρνητικές τιμές είναι ουσιαστική.

ΠΑΡΑΔΕΙΓΜΑ ΡΥ07, Όγκος και επιφάνεια ενός κουτιού

```
length = 35
width = 25
height = 8
volume = length * width * height
surface = 2 * (length * width + length * height + width * height)
print("Όγκος:", volume, "cm³")
print("Επιφάνεια:", surface, "cm²")
```

Έξοδος της συγκεκριμένης δοκιμής

```
Όγκος: 7000 cm³
Επιφάνεια: 2350 cm²
```

Οι τρεις διαστάσεις χρησιμοποιούνται με σταθερές μονάδες σε εκατοστά. Ο όγκος προκύπτει σε cm^3 και η επιφάνεια σε cm^2 . Η σαφής αναγραφή μονάδων αποτρέπει λάθη όταν αργότερα προσθέσουμε κόστος υλικού.

Οι float αποθηκεύουν πολλές δεκαδικές τιμές προσεγγιστικά. Η $0.1+0.2$ μπορεί να μην είναι ακριβώς ίση με 0.3 . Η math.isclose συγκρίνει με σχετική και προαιρετική απόλυτη ανοχή. Κοντά στο μηδέν έχει σημασία η abs_tol . Δεν στρογγυλοποιούμε κάθε ενδιάμεσο βήμα χωρίς λόγο.

ΠΑΡΑΔΕΙΓΜΑ ΡΥ08, Έλεγχος προσεγγιστικής ισότητας

```
# Η σύγκριση γίνεται με μικρή αριθμητική ανοχή. Δεν διορθώνουμε τον
# αριθμό αλλά ορίζουμε τι απόκλιση δεχόμαστε.
import math
print(0.1 + 0.2 == 0.3)
print(math.isclose(0.1 + 0.2, 0.3, abs_tol=1e-12))
print(math.isclose(math.sin(math.pi), 0.0, abs_tol=1e-12))
```

Έξοδος της συγκεκριμένης δοκιμής

```
False
True
True
```

Η ανοχή εκφράζει πόση διαφορά θεωρούμε αποδεκτή για τον συγκεκριμένο έλεγχο. Δεν αποδεικνύει μαθηματική ταυτότητα.

Δοκιμάστε πριν διαβάσετε την απάντηση, Τι σφάλμα γίνεται αν πολλαπλασιάσουμε δεύτερη φορά τον όγκο με το ύψος;

Απάντηση και αιτιολόγηση, Το ύψος έχει ήδη συνυπολογιστεί στον τύπο μήκος × πλάτος × ύψος. Αν το πολλαπλασιάσουμε ξανά, δεν υπολογίζουμε πλέον όγκο του ίδιου κουτιού αλλά διαφορετική ποσότητα.

Τεκμηρίωση κεφαλαίου: [P2], [P9], [P10].

05 | Συνθήκες και λογική

Στόχος, Να διακρίνετε περιπτώσεις, να ορίζετε σωστά διαστήματα και να χειρίζεστε την απουσία αποτελέσματος.

Το if εκτελεί ένα μπλοκ όταν η συνθήκη είναι αληθής. Το elif εξετάζει άλλη περίπτωση μόνο όταν οι προηγούμενες απέτυχαν. Το else καλύπτει όσα απέμειναν. Η άνω και κάτω τελεία κλείνει τη γραμμή της συνθήκης και οι εσοχές δηλώνουν ποιες εντολές ανήκουν στο σώμα.

ΠΑΡΑΔΕΙΓΜΑ PY09, Έλεγχος βαθμού και χαρακτηρισμός

```
# Τα όρια εξετάζονται από το υψηλότερο προς το χαμηλότερο. Έτσι
# επιλέγεται μία κατηγορία και όχι πολλές.
grade = 6.5
if not 0 <= grade <= 10:
    print("Μη έγκυρος βαθμός.")
elif grade >= 8.5:
    print("Άριστα")
elif grade >= 6.5:
    print("Λίαν Καλώς")
elif grade >= 5:
    print("Καλώς")
else:
    print("Αποτυχία")
```

Έξοδος της συγκεκριμένης δοκιμής

Λίαν Καλώς

Η Python επιτρέπει την αλυσιδωτή σύγκριση `0 <= grade <= 10`. Δεν μεταφέρουμε αυτή τη γραφή αυτούσια στη JavaScript. Οι οριακές τιμές ανήκουν σε μία μόνο κατηγορία.

Εργαλείο	Παράδειγμα
and	price <= budget and available
or	choice == 1 or choice == 2
not	not found
== και !=	Σύγκριση τιμών.
is None	Έλεγχος ότι δεν έχει οριστεί αποτέλεσμα.
in	code in available_codes

ΠΑΡΑΔΕΙΓΜΑ PY10, Μηδέν και None είναι διαφορετικά

```
# Το None σημαίνει απουσία τιμής. Η αριθμητική τιμή 0 μπορεί να είναι
# απολύτως έγκυρο αποτέλεσμα.
result = 0
if result is None:
    print("Δεν υπάρχει αποτέλεσμα.")
else:
    print("Αποτέλεσμα:", result)
print(bool([]))
print(bool("0"))
```

Έξοδος της συγκεκριμένης δοκιμής

Αποτέλεσμα: 0
False
True

Η κενή λίστα είναι ψευδής σε λογικό έλεγχο. Το μη κενό κείμενο "0" είναι αληθές. Για προαιρετικό αριθμητικό αποτέλεσμα χρησιμοποιούμε ρητό is None ώστε να μη χαθεί το νόμιμο μηδέν.

Όταν περισσότερες από μία προειδοποιήσεις μπορούν να ισχύουν μαζί, χρειαζόμαστε ανεξάρτητα if. Για παράδειγμα, μια εγγραφή μπορεί να έχει και μεγάλη ισχύ και πολλές ώρες λειτουργίας. Το elif θα επέλεγε μόνο μία από τις δύο προειδοποιήσεις.

Όρια πριν από παραδείγματα, Δοκιμάστε ακριβώς το κατώτερο και το ανώτερο όριο και μια τιμή λίγο έξω από καθένα. Το «φαίνεται σωστό για 7» δεν αποδεικνύει ότι ο έλεγχος του 0 ή του 10 έχει γραφτεί σωστά.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ποιο είναι το λάθος στο if choice == 1 or 2;

Απάντηση και αιτιολόγηση, Το 2 είναι μη μηδενική τιμή και θεωρείται αληθές. Η σωστή γραφή είναι choice == 1 or choice == 2 ή choice in (1, 2).

Τεκμηρίωση κεφαλαίου: [P3], [P19].

06 | for, range, while και παρακολούθηση βημάτων

Στόχος, Να διαλέγετε επανάληψη με βάση το πρόβλημα και να εξηγείτε πότε και γιατί σταματά.

Η for περνά από στοιχεία. Η range παράγει ακολουθία ακεραίων που χρησιμοποιούμε συχνά ως μετρητή. Το τελικό όριο δεν περιλαμβάνεται: range(1,4) δίνει 1,2,3. Η while επαναλαμβάνει όσο μια συνθήκη ισχύει. Μια while χρειάζεται σαφή τρόπο τερματισμού.

ΠΑΡΑΔΕΙΓΜΑ PY11, Άθροισμα και αριθμός βήματος

```
# Ο βρόχος δίνει μία τιμή κάθε φορά. Το άθροισμα πρέπει να ξεκινά πριν
# από τον βρόχο ώστε να μη μηδενίζεται ξανά.
total = 0
for number in range(1, 5):
    total += number
    print(number, total)
```

Έξοδος της συγκεκριμένης δοκιμής

1 1
2 3
3 6
4 10

Το 5 δεν συμμετέχει στη range. Ο μετρητής παίρνει 1,2,3,4 και το άθροισμα εξελίσσεται σε 1,3,6,10.

ΠΑΡΑΔΕΙΓΜΑ PY12, Αρίθμηση για τον χρήστη, θέση για τον κώδικα

```
# Η αρίθμηση παρουσίασης αρχίζει από 1. Δεν αλλάζει η εσωτερική αρίθμηση
# των θέσεων της λίστας από 0.
grades = [6, 8, 10]
for student_number, grade in enumerate(grades, start=1):
    print(f"Φοιτητής {student_number}: {grade}")
```

Έξοδος της συγκεκριμένης δοκιμής

Φοιτητής 1: 6
Φοιτητής 2: 8
Φοιτητής 3: 10

Η enumerate επιστρέφει ζεύγος αρίθμησης και τιμής. Το start=1 αφορά την εμφανιζόμενη αρίθμηση· δεν αλλάζει τις πραγματικές θέσεις της λίστας.

ΠΑΡΑΔΕΙΓΜΑ ΡΥ13, Μενού που τερματίζει με συγκεκριμένη επιλογή

Η επιλογή μενού παραμένει κείμενο. Η break τερματίζει τον βρόχο, ενώ η
continue παραλείπει μόνο τον τρέχοντα γύρο.

while True:

```
choice = input("1 Χαιρετισμός, 0 Έξοδος: ").strip()
if choice == "0":
    break
if choice != "1":
    print("Μη έγκυρη επιλογή.")
    continue
print("Καλή μελέτη!")
print("Τέλος.")
```

Δοκιμή εισόδου, με τη σειρά: 2 → 1 → 0.

Έξοδος της συγκεκριμένης δοκιμής

Μη έγκυρη επιλογή.
Καλή μελέτη!
Τέλος.

Η break σταματά τον βρόχο. Η continue παραλείπει τον υπόλοιπο γύρο. Στο παράδειγμα δεν μετατρέπουμε το μενού σε αριθμό, αφού συγκρίνουμε μόνο κωδικούς κειμένου.

Σε ένθετες επαναλήψεις, η break αφορά την πλησιέστερη επανάληψη. Αν θέλουμε να σταματήσει ολόκληρη μια λειτουργία που περιέχει πολλούς βρόχους, μια συνάρτηση με return συχνά δίνει καθαρότερη δομή. Μην αλλάζετε το μήκος μιας λίστας ενώ περνάτε από την ίδια χωρίς να έχετε προβλέψει την επίδραση στις θέσεις.

Δοκιμάστε πριν διαβάσετε την απάντηση, Πόσα στοιχεία έχει η list(range(2, 9, 2));

Απάντηση και αιτιολόγηση, Έχει τέσσερα στοιχεία: [2,4,6,8]. Το 9 είναι αποκλειστικό άνω όριο και το βήμα είναι 2.

Τεκμηρίωση κεφαλαίου: [P3].

07 | Συναρτήσεις με καθαρές ευθύνες

Στόχος, Να σχεδιάζετε παραμέτρους, επιστρεφόμενα αποτελέσματα και τεκμηρίωση χωρίς περιττές εξαρτήσεις.

Η def ορίζει συνάρτηση. Η κλήση την εκτελεί. Ο κώδικας μιας συνάρτησης δεν εκτελείται απλώς επειδή διαβάστηκε ο ορισμός της. Η return επιστρέφει αποτέλεσμα στον καλούντα και σταματά τη συγκεκριμένη κλήση. Χωρίς ρητή return επιστρέφεται None.

ΠΑΡΑΔΕΙΓΜΑ ΡΥ14, Υπολογισμός χωριστά από την εμφάνιση

```
def discounted_price(price, percent):
    """Επιστρέφει τιμή προσφοράς χωρίς αλλαγή της λιανικής."""
    return price * (1 - percent / 100)

result = discounted_price(80, 25)
print(result)
print(discounted_price(120, 10))
```

Έξοδος της συγκεκριμένης δοκιμής

60.0
108.0

Οι price και percent είναι παράμετροι. Τα 80 και 25 είναι ορίσματα. Η docstring, το πρώτο κείμενο μέσα στη συνάρτηση, εξηγεί τη λειτουργία της και είναι διαθέσιμη ως τεκμηρίωση.

ΠΑΡΑΔΕΙΓΜΑ PY15, Περισσότερα από ένα αποτελέσματα

```
def summarize(values):
    """Δέχεται μη κενή λίστα και δίνει άθροισμα και μέσο όρο."""
    if not values:
        raise ValueError("Χρειάζεται τουλάχιστον μία τιμή.")
    total = sum(values)
    return total, total / len(values)
total, average = summarize([4, 8, 9])
print(total, average)
```

Έξοδος της συγκεκριμένης δοκιμής

```
21 7.0
```

Η επιστροφή είναι μία πλειάδα δύο τιμών. Η ανάθεση την αποσυσκευάζει σε δύο ονόματα. Αν δεν χρειάζεται αποτέλεσμα, μια συνάρτηση αναφοράς μπορεί να επιστρέφει None.

Προεπιλεγμένες παράμετροι με προσοχή

Μια προεπιλεγμένη τιμή χρησιμοποιείται όταν παραλείπεται το αντίστοιχο όρισμα. Για μεταβαλλόμενες συλλογές αποφεύγουμε `def f(items=[])`, επειδή η ίδια λίστα μπορεί να χρησιμοποιηθεί σε πολλές κλήσεις. Δίνουμε None και δημιουργούμε νέα λίστα μέσα στη συνάρτηση.

ΠΑΡΑΔΕΙΓΜΑ PY16, Νέα λίστα για κάθε ανεξάρτητη κλήση

```
def add_note(text, notes=None):
    """Προσθέτει σε δοσμένη λίστα ή δημιουργεί καινούργια."""
    if notes is None:
        notes = []
    notes.append(text)
    return notes
print(add_note("Πρώτη"))
print(add_note("Δεύτερη"))
```

Έξοδος της συγκεκριμένης δοκιμής

```
['Πρώτη']
['Δεύτερη']
```

Οι δύο εκτυπώσεις δεν μοιράζονται κρυφά μία προεπιλεγμένη λίστα. Αν δοθεί ρητά μια λίστα ως όρισμα, η συνάρτηση την τροποποιεί και αυτό αναφέρεται στην τεκμηρίωση.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί μια συνάρτηση volume πρέπει συνήθως να επιστρέφει τον αριθμό αντί να τον τυπώνει μόνο;

Απάντηση και αιτιολόγηση, Για να μπορεί το πρόγραμμα να συγκρίνει όγκους, να επιλέξει κουτί, να αποθηκεύσει αναφορά ή να ελέγξει το αποτέλεσμα χωρίς να επεξεργάζεται κείμενο από την οθόνη.

Τεκμηρίωση κεφαλαίου: [P3], [P7].

08 | Λίστες, θέσεις και πραγματικά αντίγραφα

Στόχος, Να αλλάζετε μια συλλογή με ασφάλεια και να καταλαβαίνετε πότε δύο ονόματα μοιράζονται δεδομένα.

Η λίστα κρατά στοιχεία με σειρά και επιτρέπει αλλαγές. Η πρώτη θέση είναι 0 και η τελευταία μπορεί να διαβαστεί ως -1. Η έκφραση `values[1:3]` δίνει νέο τμήμα με θέσεις 1 και 2· η θέση 3 δεν περιλαμβάνεται. Μια θέση έξω από τα όρια προκαλεί `IndexError`, δεν επιστρέφει αυτόματα κενό.

ΠΑΡΑΔΕΙΓΜΑ PY17, Αλλαγή, προσθήκη και αφαίρεση

```
grades = [6, 8, 10]
# Διόρθωση δευτέρου φοιτητή: δεν δημιουργούμε νέα εγγραφή.
grades[1] = 9
grades.append(7)
print(grades)
print(grades[1:3])
removed = grades.pop()
print("Αφαιρέθηκε:", removed)
print("Πλήθος:", len(grades))
```

Έξοδος της συγκεκριμένης δοκιμής

```
[6, 9, 10, 7]
[9, 10]
Αφαιρέθηκε: 7
Πλήθος: 3
```

Η append προσθέτει στο τέλος. Η pop χωρίς θέση αφαιρεί το τελευταίο στοιχείο. Η remove(value) αφαιρεί την πρώτη εμφάνιση συγκεκριμένης τιμής, όχι συγκεκριμένη θέση.

ΠΑΡΑΔΕΙΓΜΑ PY18, Ανάθεση και ρηχό αντίγραφο

```
# Η απλή ανάθεση μοιράζεται την ίδια λίστα. Η copy δημιουργεί άλλη
# εξωτερική λίστα, όχι αυτόματα βαθιά αντίγραφα.
original = [4, 6]
same = original
copy = original.copy()
same[0] = 9
copy[1] = 8
print(original)
print(copy)
```

Έξοδος της συγκεκριμένης δοκιμής

```
[9, 6]
[4, 8]
```

Το same και το original αναφέρονται στην ίδια λίστα. Η copy δημιουργεί νέο εξωτερικό αντικείμενο. Η λέξη «εξωτερικό» είναι σημαντική όταν η λίστα περιέχει λεξικά ή άλλες λίστες.

ΠΑΡΑΔΕΙΓΜΑ PY19, Αντίγραφα εγγραφών για ανεξάρτητο σενάριο

```
# Αντιγράφουμε κάθε λεξικό χωριστά. Έτσι το δοκιμαστικό σενάριο δεν
# μεταβάλλει τα αρχικά δεδομένα.
devices = [{"name": "Υπολογιστές", "hours": 5}]
trial = []
for device in devices:
    trial.append(device.copy())
trial[0]["hours"] = 4
print("Αρχικές ώρες:", devices[0]["hours"])
print("Ωρες σεναρίου:", trial[0]["hours"])
```

Έξοδος της συγκεκριμένης δοκιμής

```
Αρχικές ώρες: 5
Ωρες σεναρίου: 4
```

Αν κάναμε μόνο trial = devices.copy(), τα δύο εξωτερικά αντικείμενα θα περιείχαν αναφορά στο ίδιο λεξικό. Η ξεχωριστή copy κάθε εγγραφής αρκεί εδώ επειδή οι τιμές της είναι κείμενα και αριθμοί, όχι βαθύτερες μεταβαλλόμενες δομές.

Μέθοδοι που αλλάζουν τη λίστα, Η append και η sort μεταβάλλουν τη λίστα και επιστρέφουν None. Δεν γράφουμε grades = grades.sort(). Για νέο ταξινομημένο αντικείμενο χρησιμοποιούμε sorted(grades).

Δοκιμάστε πριν διαβάσετε την απάντηση, Μετά από δεκαπέντε καταχωρίσεις, πώς διορθώνουμε τον φοιτητή με αριθμό 5 χωρίς να προσθέσουμε 16η εγγραφή;

Απάντηση και αιτιολόγηση, Ελέγχουμε ότι ο αριθμός είναι από 1 έως 15 και κάνουμε `grades[5 - 1] = new_grade`. Η `append` θα ήταν λανθασμένη, γιατί θα πρόσθετε νέο στοιχείο.

Τεκμηρίωση κεφαλαίου: [P4].

09 | Στατιστικά, φίλτρα και ταξινόμηση

Στόχος, Να μετατρέπετε μια λίστα σε αναφορά και να διατηρείτε ισοβαθμίες και αρχική σειρά όπου απαιτείται.

Η `sum` υπολογίζει άθροισμα και η `len` πλήθος. Ο μέσος όρος χρειάζεται μη κενή λίστα. Η `min` και η `max` δίνουν ακραίες τιμές, αλλά δεν δίνουν αυτόματα όλους τους ανθρώπους ή όλες τις συσκευές που μοιράζονται εκείνη την τιμή. Για ισοβαθμίες χρειαζόμαστε ξεχωριστό πέρασμα.

ΠΑΡΑΔΕΙΓΜΑ PY20, Δύο περάσματα για μία σωστή σύνοψη

```
# Η επιλογή των τιμών πάνω από τον μέσο χρειάζεται δεύτερο πέρασμα: πρώτα
# πρέπει να γνωρίζουμε τον τελικό μέσο.
grades = [9, 9, 6, 4]
average = sum(grades) / len(grades)
maximum = max(grades)
above = []
top = []
for number, grade in enumerate(grades, 1):
    if grade > average:
        above.append(number)
    if grade == maximum:
        top.append(number)
print("Μέσος:", average)
print("Πάνω από τον μέσο:", above)
print("Κορυφαίοι:", top)
```

Έξοδος της συγκεκριμένης δοκιμής

```
Μέσος: 7.0
Πάνω από τον μέσο: [1, 2]
Κορυφαίοι: [1, 2]
```

Συγκρίνουμε τις πραγματικές τιμές και όχι την εμφάνιση δύο δεκαδικών. Οι αριθμοί φοιτητών αρχίζουν από 1 για τον χρήστη.

Προηγείται ο αναλυτικός βρόχος

Η συνοπτική κατασκευή λίστας (list comprehension) γράφει συνοπτικά την κατασκευή νέας λίστας. Η `[g for g in grades if g >= 5]` αντιστοιχεί σε `for`, `if` και `append`. Δεν είναι αναγκαίο να τη χρησιμοποιήσετε από την πρώτη ημέρα. Προτιμήστε τη μορφή που μπορείτε να εξηγήσετε με ακρίβεια.

ΠΑΡΑΔΕΙΓΜΑ PY21, Φιλτράρισμα και ταξινόμηση με κριτήριο

```
# Το φίλτρο επιλέγει εγγραφές και η sorted δίνει νέα σειρά. Οι εγγραφές
# με ίδιο κριτήριο διατηρούν τη μεταξύ τους σειρά.
phones = [
    {"model": "A", "price": 450},
    {"model": "B", "price": 300},
    {"model": "C", "price": 450},
]
matches = [phone for phone in phones if phone["price"] <= 450]
ordered = sorted(matches, key=lambda phone: phone["price"])
for phone in ordered:
    print(phone["model"], phone["price"])
```

Έξοδος της συγκεκριμένης δοκιμής

```
B 300
```

A 450
C 450

Η key δίνει την τιμή σύγκρισης κάθε εγγραφής. Η lambda είναι μικρή ανώνυμη συνάρτηση. Οι ίσες τιμές διατηρούν τη μεταξύ τους σειρά. Η sorted επιστρέφει νέο εξωτερικό κατάλογο, αλλά δεν αντιγράφει βαθιά τις ίδιες εγγραφές.

ΠΑΡΑΔΕΙΓΜΑ PY22, Διάμεσος χωρίς αλλαγή αρχικής λίστας

```
def median(values):
    """Επιστρέφει τη μεσαία τιμή ή τον μέσο των δύο μεσαίων."""
    if not values:
        raise ValueError("Δεν υπάρχουν τιμές.")
    ordered = sorted(values)
    middle = len(ordered) // 2
    if len(ordered) % 2 == 1:
        return ordered[middle]
    return (ordered[middle - 1] + ordered[middle]) / 2

print(median([10, 2, 6]))
print(median([10, 2, 6, 8]))
```

Έξοδος της συγκεκριμένης δοκιμής

6
7.0

Ο μέσος όρος και η διάμεσος απαντούν διαφορετικά ερωτήματα. Στη διάμεσο προηγείται πάντα η ταξινόμηση.

Δοκιμάστε πριν διαβάσετε την απάντηση, Αν όλοι οι βαθμοί είναι 5, πόσοι είναι πάνω από τον μέσο;

Απάντηση και αιτιολόγηση, Κανένας. Η ισότητα δεν είναι αυστηρά μεγαλύτερο. Όλοι όμως μοιράζονται τον μέγιστο βαθμό και όλοι είναι επιτυχείς όταν το όριο είναι 5.

Τεκμηρίωση κεφαλαίου: [P4], [P15].

10 | Λεξικά, πλειάδες και σύνολα

Στόχος, Να κρατάτε οργανωμένες εγγραφές, ζεύγη θέσεων και μοναδικά στοιχεία χωρίς να μπερδεύετε τις δομές.

Η επιλογή δομής πρέπει να ακολουθεί το πρόβλημα. Μια λίστα είναι χρήσιμη για σειρά εγγραφών. Ένα λεξικό αντιστοιχίζει κλειδιά σε τιμές. Μια πλειάδα κρατά σταθερό συνδυασμό τιμών, όπως (γραμμή, στήλη). Ένα σύνολο χρησιμοποιείται όταν μας ενδιαφέρει η μοναδικότητα και ο έλεγχος συμμετοχής, όχι μια εγγυημένη σειρά εμφάνισης.

Δομή	Παράδειγμα	Συνηθισμένη χρήση
list	[6, 8, 10]	Βαθμοί με σειρά.
dict	{"name": "A1", "watts": 20}	Ονομασμένα πεδία μιας συσκευής.
tuple	(2, 3)	Μία θέση σε χάρτη.
set	{"A01", "B02"}	Κωδικοί που έχουν ήδη χρησιμοποιηθεί.

ΠΑΡΑΔΕΙΓΜΑ PY23, Εγγραφή και ασφαλής ανάγνωση

```
# Η πρόσβαση με get επιτρέπει προεπιλεγμένη τιμή όταν λείπει ένα κλειδί.
# Δεν δημιουργεί αυτόματα νέα ιδιότητα.
device = {"name": "A1", "watts": 20, "hours": 5}
print(device["name"])
print(device.get("quantity", 1))
print("quantity" in device)
```

```
for key, value in device.items():
    print(key, "=", value)
```

Έξοδος της συγκεκριμένης δοκιμής

```
A1
1
False
name = A1
watts = 20
hours = 5
```

Η πρόσβαση με αγκύλες σε ανύπαρκτο κλειδί προκαλεί `KeyError`. Η `get` επιτρέπει προεπιλογή, αλλά δεν πρέπει να χρησιμοποιείται για να κρύψει την απουσία πεδίου που είναι υποχρεωτικό.

ΠΑΡΑΔΕΙΓΜΑ PY24, Θέσεις και γνωστά κελιά

```
# Η πλειάδα κρατά μια θέση ως ενιαία τιμή. Το σύνολο θυμάται ποιες θέσεις
# έχουν ήδη εξεταστεί.
start = (0, 0)
visited = {start}
visited.add((0, 1))
visited.add((0, 1))
print(len(visited))
print((0, 1) in visited)
row, column = start
print(row, column)
```

Έξοδος της συγκεκριμένης δοκιμής

```
2
True
0 0
```

Η ίδια θέση προστίθεται μία φορά σε σύνολο. Μια πλειάδα ακεραίων μπορεί να χρησιμοποιηθεί και ως κλειδί λεξικού. Μια μεταβαλλόμενη λίστα δεν χρησιμοποιείται έτσι.

Η ανάθεση `device["hours"] = 4` αλλάζει υπάρχον πεδίο. Η νέα ανάθεση σε ένα κλειδί που υπήρχε ήδη αντικαθιστά την προηγούμενη τιμή του. Όταν καταχωρίζουμε κωδικούς, ελέγχουμε πρώτα τη συμμετοχή ώστε μια δεύτερη εγγραφή να μην αντικαταστήσει σιωπηρά την πρώτη.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ποια δομή ταιριάζει στο `parent` ενός λαβυρίνθου;

Απάντηση και αιτιολόγηση, Λεξικό που αντιστοιχίζει κάθε θέση στην προηγούμενη θέση από την οποία τη βρήκαμε. Έτσι η θέση λειτουργεί ως κλειδί και η προηγούμενη θέση ως τιμή.

Τεκμηρίωση κεφαλαίου: [P4], [P19].

11 | Ανθεκτική είσοδος και εξαιρέσεις

Στόχος, Να επαναλαμβάνετε μόνο αναμενόμενα λάθη και να επιτρέπετε ομαλή διακοπή της εκτέλεσης.

Μια μετατροπή `float` μπορεί να αποτύχει με `ValueError`. Μπορεί όμως και να πετύχει για `"nan"` ή `"inf"`, που δεν θέλουμε ως συνηθισμένα δεδομένα. Άρα χρειάζεται και `math.isfinite`. Η εγκυρότητα δεν εξαντλείται στο ότι δεν εμφανίστηκε εξαίρεση.

ΠΑΡΑΔΕΙΓΜΑ PY25, Αναγνώστης αριθμού με όρια

```
import math

def read_number(message, minimum, maximum):
    """Επαναλαμβάνει άκυρες τιμές, αλλά δεν παγιδεύει τη διακοπή."""
    # Επαναλαμβάνουμε μόνο αναμενόμενα λάθη τιμής, όχι τη διακοπή του
    # χρήστη.
    while True:
```

```

raw = input(message).strip().replace(",", ".")
try:
    value = float(raw)
except ValueError:
    print("Δώστε αριθμό, όχι κενό ή γράμματα.")
    continue
if not math.isfinite(value):
    print("Η τιμή πρέπει να είναι πεπερασμένη.")
    continue
if not minimum <= value <= maximum:
    print(f"Όρια: {minimum} έως {maximum}.")
    continue
return value

try:
    grade = read_number("Βαθμός 0-10: ", 0, 10)
    print("Αποδεκτός:", grade)
except (EOFError, KeyboardInterrupt):
    print("Η εκτέλεση διακόπηκε.")

```

Δοκιμή εισόδου, με τη σειρά: abc → nan → 11 → 8,5.

Έξοδος της συγκεκριμένης δοκιμής

Δώστε αριθμό, όχι κενό ή γράμματα.
 Η τιμή πρέπει να είναι πεπερασμένη.
 Όρια: 0 έως 10.
 Αποδεκτός: 8.5

Δοκιμή με abc, nan, 11 και τέλος 8,5. Οι δύο εξαιρέσεις διακοπής δεν συλλαμβάνονται μέσα στον βρόχο, επομένως δεν μετατρέπονται σε ατέρμονη νέα ερώτηση.

Η παραπάνω float δέχεται την αριθμητική σύνταξη της Python, όπως 1e2. Αν μια εφαρμογή ζητά μόνο απλή δεκαδική γραφή, προσθέτουμε έλεγχο χαρακτήρων πριν από τη μετατροπή. Για ακέραια πεδία χρησιμοποιούμε int(raw) και κατάλληλο έλεγχο ορίων, όχι float και έπειτα σιωπηρή αποκοπή.

ΠΑΡΑΔΕΙΓΜΑ PY26, Έλεγχος τιμής και όταν καλείται συνάρτηση από άλλο κώδικα

```

import math

def require_price(value):
    """Απορρίπτει και λογικές τιμές, όχι μόνο αρνητικούς αριθμούς."""
    if type(value) not in (int, float):
        raise TypeError("Η τιμή πρέπει να είναι αριθμός.")
    if not math.isfinite(value) or not 0 < value <= 10000:
        raise ValueError("Η τιμή πρέπει να είναι από 0 έως 10000, όχι 0.")
    return float(value)

try:
    require_price(True)
except (TypeError, ValueError) as error:
    print(error)
print(require_price(450))

```

Έξοδος της συγκεκριμένης δοκιμής

Η τιμή πρέπει να είναι αριθμός.
 450.0

Στην Python οι bool είναι υποτύπος των int. Ένας απλός isinstance(value, int) δέχεται και το True. Για τιμές όπως πλήθος συσκευών ή λιανική, επιλέγουμε ρητά αν αυτό επιτρέπεται. Εδώ δεν επιτρέπεται.

Όχι γενικό except χωρίς λόγο, Ένα σκέτο except μπορεί να κρύψει λάθη του προγραμματιστή και τη διακοπή του χρήστη. Συλλαμβάνουμε μόνο όσα περιμένουμε και μπορούμε να αντιμετωπίσουμε. Για κάθε catch/except πρέπει να εξηγούμε ποια αποτυχία χειρίζεται.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί ένας setter χρειάζεται έλεγχο ακόμη κι αν υπάρχει αναγνώστης εισόδου;

Απάντηση και αιτιολόγηση, Το αντικείμενο μπορεί να αλλάξει και από άλλο τμήμα κώδικα, όχι μόνο από πληκτρολόγηση. Ο setter προστατεύει τον κανόνα σε κάθε διαδρομή αλλαγής.

Τεκμηρίωση κεφαλαίου: [P6], [P10], [P15].

12 | Άρθρωμα, math, random και οργάνωση αρχείου

Στόχος, Να χρησιμοποιείτε έτοιμα εργαλεία χωρίς να κρύβετε τη ροή του προγράμματος και να φτιάχνετε ελέγξιμες προσομοιώσεις.

Ένα άρθρωμα (module) είναι αρχείο ή οργανωμένη μονάδα κώδικα που μπορούμε να εισαγάγουμε. Η `import math` κρατά ορατή την προέλευση: `math.sqrt`. Η `from random import Random` εισάγει ένα συγκεκριμένο όνομα. Αποφεύγουμε γενικά `from ... import *`, γιατί δυσκολεύει να καταλάβουμε από πού προήλθαν τα ονόματα.

ΠΑΡΑΔΕΙΓΜΑ PY27, Μαθηματικά εργαλεία και μετατροπή γωνιών

```
import math

def sine_degrees(degrees):
    """Η είσοδος είναι σε μοίρες, η math.sin χρειάζεται ακτίνια."""
    return math.sin(math.radians(degrees))

print(f"sin(30°) = {sine_degrees(30):.6f}")
print(math.hypot(3, 4))
print(math.floor(3.9))
```

Έξοδος της συγκεκριμένης δοκιμής

```
sin(30°) = 0.500000
5.0
3
```

Πριν από `math.sqrt` ελέγχουμε μη αρνητικό αριθμό. Πριν από λογάριθμο ελέγχουμε θετικό αριθμό. Ο έλεγχος της εισόδου ανήκει στους κανόνες της εφαρμογής και δεν πρέπει να παραλείπεται επειδή υπάρχει έτοιμη συνάρτηση.

ΠΑΡΑΔΕΙΓΜΑ PY28, Τυχαιότητα που μπορούμε να αναπαράγουμε στη δοκιμή

```
from random import Random

rng = Random(17)
# Το τοπικό αντικείμενο δεν αλλάζει τον γενικό τυχαίο μηχανισμό.
readings = [rng.randint(18, 28) for _ in range(5)]
print(readings)
second_rng = Random(17)
print(readings == [second_rng.randint(18, 28) for _ in range(5)])
```

Έξοδος της συγκεκριμένης δοκιμής

```
[26, 24, 22, 23, 22]
True
```

Ίδιο seed και ίδια σειρά κλήσεων, στο ίδιο περιβάλλον, επιτρέπουν αναπαραγωγή της δοκιμής. Δεν υποθέτουμε ότι κάθε βοηθητική μέθοδος δίνει αιώνια ίδια αποτελέσματα σε όλες τις εκδόσεις. Η `randint(a,b)` περιλαμβάνει και τα δύο άκρα.

Η main και η προστασία εκκίνησης

ΠΑΡΑΔΕΙΓΜΑ PY29, Αρχείο που μπορεί και να εκτελεστεί και να εισαχθεί

```
def to_liters(cubic_meters):
    """Επιστρέφει λίτρα για έναν όγκο σε κυβικά μέτρα."""
    return cubic_meters * 1000
```

```
def main():
    print(to_liters(1.5))
if __name__ == "__main__":
    main()
```

Έξοδος της συγκεκριμένης δοκιμής

1500.0

Η συνάρτηση main δεν έχει ειδική σημασία από μόνη της. Ο τελευταίος έλεγχος την καλεί μόνο όταν το αρχείο εκτελείται ως κύριο πρόγραμμα. Έτσι οι δοκιμές μπορούν να εισάγουν συναρτήσεις χωρίς να εμφανίζονται αυτόματα ερωτήσεις input.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί δεν πρέπει να ονομάσουμε το δικό μας αρχείο random.py όταν θέλουμε import random;

Απάντηση και αιτιολόγηση, Το δικό μας αρχείο μπορεί να καλύψει το άρθρωμα της βιβλιοθήκης και να προκαλέσει σύγχυση στις εισαγωγές. Επιλέγουμε περιγραφικά ονόματα που δεν συγκρούονται με τα εργαλεία που χρησιμοποιούμε.

Τεκμηρίωση κεφαλαίου: [P8], [P10], [P11].

13 | Αρχεία κειμένου, JSON και CSV

Στόχος, Να αποθηκεύετε αναφορές με ελληνικά, να χειρίζεστε διαδρομές και να μην αντικαθιστάτε αρχεία χωρίς πρόθεση.

Τα δεδομένα μνήμης χάνονται όταν ολοκληρωθεί το πρόγραμμα, εκτός αν αποθηκευτούν. Η open ανοίγει αρχείο με συγκεκριμένο τρόπο. Το with εξασφαλίζει ότι το αρχείο θα κλείσει όταν φύγουμε από το μπλοκ, ακόμη και αν συμβεί σφάλμα. Το encoding="utf-8" δηλώνει πώς αποθηκεύονται τα ελληνικά.

Τρόπος	Συμπεριφορά
"r"	Ανάγνωση υπάρχοντος αρχείου.
"w"	Εγγραφή: αντικαθιστά υπάρχον περιεχόμενο.
"a"	Προσθήκη στο τέλος.
"x"	Δημιουργία μόνο νέου αρχείου· αποτυχία αν υπάρχει ήδη.

ΠΑΡΑΔΕΙΓΜΑ PY30, Αναφορά που δεν αντικαθιστά υπάρχον αρχείο

```
# Η λειτουργία x δημιουργεί μόνο νέο αρχείο. Αν υπάρχει αρχείο με το ίδιο
# όνομα, χειριζόμαστε το λάθος χωρίς αντικατάσταση.
lines = ["Αναφορά βαθμολογίας", "Μέσος όρος: 7.25"]
text = "\n".join(lines)
print(text)
try:
    with open("notes_report.txt", "x", encoding="utf-8") as file:
        file.write(text + "\n")
    print("Το νέο αρχείο δημιουργήθηκε.")
except FileExistsError:
    print("Το αρχείο υπάρχει ήδη και δεν αντικαταστάθηκε.")
except OSError as error:
    print("Δεν έγινε αποθήκευση:", error)
```

Έξοδος της συγκεκριμένης δοκιμής

Αναφορά βαθμολογίας
Μέσος όρος: 7.25
Το νέο αρχείο δημιουργήθηκε.

Η εμφανιζόμενη έξοδος αντιστοιχεί στην πρώτη εκτέλεση σε φάκελο όπου δεν υπάρχει το notes_report.txt. Τη δεύτερη φορά ενεργοποιείται η FileNotFoundError. Χρησιμοποιούμε το ίδιο text για οθόνη και αρχείο.

Μια σχετική διαδρομή υπολογίζεται από τον τρέχοντα φάκελο εργασίας, όχι αναγκαστικά από τον φάκελο όπου βρίσκεται ο κώδικας. Η Path.cwd() δείχνει πού εργαζόμαστε. Μην μαντεύετε πού γράφτηκε ένα αρχείο όταν εκτελείτε από διαφορετικό IDE. Προτιμάτε σαφές μήνυμα που αναφέρει τη διαδρομή.

ΠΑΡΑΔΕΙΓΜΑ PY31, Κείμενο JSON και δεδομένα στη μνήμη

```
# Η JSON μετατρέπει δομές δεδομένων σε κείμενο και αντίστροφα. Η
# αποκωδικοποίηση δεν επαληθεύει τους κανόνες της εφαρμογής.
import json

original = {"κατασκευαστής": "Demo", "μοντέλο": "A1", "τιμή": 499.0}
text = json.dumps(original, ensure_ascii=False)
restored = json.loads(text)
print(text)
print(restored["μοντέλο"])
print(restored == original)
```

Έξοδος της συγκεκριμένης δοκιμής

```
{"κατασκευαστής": "Demo", "μοντέλο": "A1", "τιμή": 499.0}
A1
True
```

Το JSON είναι μορφή αναπαράστασης δεδομένων. Η loads δεν δημιουργεί αυτόματα δικές μας κλάσεις και δεν ελέγχει τους κανόνες της εφαρμογής. Μετά τη φόρτωση ελέγχουμε υποχρεωτικά πεδία, τύπους και όρια πριν αντικαταστήσουμε την τρέχουσα κατάσταση.

ΠΑΡΑΔΕΙΓΜΑ PY32, CSV χωρίς χειροκίνητο διαχωρισμό πεδίων

```
# Το CSV αναγνωρίζει τα πεδία ακόμη κι όταν περιέχουν κόμμα. Το απλό
# split δεν γνωρίζει τον κανόνα των εισαγωγικών.
import csv
import io

buffer = io.StringIO()
writer = csv.writer(buffer)
writer.writerow(["όνομα", "βαθμός"])
writer.writerow(["Αννα, ομάδα Α", 8.5])
text = buffer.getvalue()
rows = list(csv.reader(io.StringIO(text)))
print(rows[1][0])
print(rows[1][1])
```

Έξοδος της συγκεκριμένης δοκιμής

```
Αννα, ομάδα Α
8.5
```

Το όνομα περιέχει κόμμα, αλλά ο csv χειρίζεται σωστά τη μορφή. Σε πραγματικό αρχείο CSV χρησιμοποιούμε open(..., newline="", encoding="utf-8"). Η ανάγνωση επιστρέφει κείμενα: ο βαθμός χρειάζεται μετατροπή και έλεγχο.

Δοκιμάστε πριν διαβάσετε την απάντηση, Αρκεί ένα επιτυχημένο json.loads για να θεωρηθεί ασφαλής η φόρτωση;

Απάντηση και αιτιολόγηση, Όχι. Μπορεί το JSON να είναι συντακτικά σωστό αλλά να λείπουν πεδία, να υπάρχουν αρνητικές τιμές ή διπλοί κωδικοί. Ελέγχουμε όλα τα νέα δεδομένα πριν αντικαταστήσουμε τα παλιά.

Τεκμηρίωση κεφαλαίου: [P5], [P13], [P17], [P18].

14 | Κλάσεις, self και ελεγχόμενες ιδιότητες

Στόχος, Να δημιουργείτε αντικείμενα με δική τους κατάσταση και να προστατεύετε τους κανόνες τους.

Μια κλάση περιγράφει κοινά χαρακτηριστικά και ενέργειες. Η κλήση `SmartPhone("Demo", "A", 500)` δημιουργεί ένα συγκεκριμένο αντικείμενο. Η `__init__` αποθηκεύει την αρχική του κατάσταση. Το `self` είναι αναφορά στο ίδιο το αντικείμενο της μεθόδου. Δύο αντικείμενα έχουν δικές τους τιμές, όχι ένα κοινό σύνολο τιμών για όλη την κλάση.

ΠΑΡΑΔΕΙΓΜΑ PY33, Τιμή με έλεγχο κατά τη δημιουργία και την αλλαγή

```
import math

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price # Καλεί τον ίδιο setter με μια μελλοντική αλλαγή.

    @property
    def price(self):
        return self.__price

    # Ο ίδιος έλεγχος προστατεύει και την αρχική δημιουργία και τις
    # μελλοντικές αναθέσεις.
    @price.setter
    def price(self, value):
        if type(value) not in (int, float):
            raise TypeError("Η τιμή πρέπει να είναι αριθμός.")
        if not math.isfinite(value) or value <= 0:
            raise ValueError("Η τιμή πρέπει να είναι θετική και πεπερασμένη.")
        self.__price = float(value)

    def discounted_price(self, percent):
        if not 0 <= percent <= 50:
            raise ValueError("Έκπτωση από 0 έως 50%.")
        # Δίνουμε νέα τιμή χωρίς να μεταβάλλουμε τη λιανική.
        return round(self.price * (1 - percent / 100), 2)

    def __str__(self):
        return f"{self.name}: {self.price:.2f} €"

product = Product("Εκπαιδευτική συσκευή", 500)
print(product)
print(product.discounted_price(10))
try:
    product.price = -10
except ValueError as error:
    print(error)
print(product.price)
```

Έξοδος της συγκεκριμένης δοκιμής

```
Εκπαιδευτική συσκευή: 500.00 €
450.0
Η τιμή πρέπει να είναι θετική και πεπερασμένη.
500.0
```

Το παράδειγμα εστιάζει στον έλεγχο της `price`. Σε πλήρη εφαρμογή ελέγχονται επιπλέον το όνομα και ο τύπος της έκπτωσης. Η τιμή δεν αλλάζει όταν ο setter απορρίψει την ανάθεση.

Με `@property` η ανάγνωση γράφεται `product.price` και όχι `product.price()`. Με `@price.setter` η ανάθεση ελέγχεται πριν αποθηκευτεί. Το όνομα `__price` υπόκειται σε μετασχηματισμό ονόματος (`name mangling`), που βοηθά στην αποφυγή ακούσιων συγκρούσεων. Δεν είναι κρυπτογράφηση και δεν αποτελεί απόλυτο φραγμό πρόσβασης όπως το `#` της JavaScript.

Στοιχείο	Ρόλος
<code>__init__</code>	Αρχικοποίηση κάθε αντικειμένου.

Στοιχείο	Ρόλος
self.__price	Τιμή που ανήκει στο συγκεκριμένο αντικείμενο.
@property	Ελεγχόμενη ανάγνωση σαν ιδιότητα.
@price.setter	Έλεγχος πριν από αποθήκευση.
__str__	Αναγνώσιμη παρουσίαση του αντικειμένου.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί το `discounted_price` επιστρέφει νέα τιμή χωρίς να γράφει στη `self.price`;

Απάντηση και αιτιολόγηση, Αν αλλάξει τη λιανική, μια δεύτερη κλήση θα εφαρμόσει έκπτωση πάνω στην ήδη μειωμένη τιμή. Η επιστροφή νέου αποτελέσματος επιτρέπει σύγκριση σεναρίων χωρίς αλλοίωση αρχικών δεδομένων.

Τεκμηρίωση κεφαλαίου: [P7], [P15].

15 | Λίστες αντικειμένων και επιλογή κατάλληλης λύσης

Στόχος, Να συνδυάζετε τις μεθόδους μιας οντότητας με αναζήτηση σε ολόκληρο κατάλογο.

Μια κλάση γνωρίζει τα στοιχεία ενός αντικειμένου. Η λίστα συγκεντρώνει πολλά τέτοια αντικείμενα. Για να βρούμε κατάλληλο κουτί, κάθε κουτί πρέπει να μπορεί να ελέγξει αν χωρά ένα προϊόν. Έπειτα μια ξεχωριστή συνάρτηση συγκρίνει μόνο τα κατάλληλα κουτιά. Ο μεγάλος όγκος από μόνος του δεν αρκεί: ένα πολύ μακρύ αλλά πολύ στενό κουτί μπορεί να μη χωρά τον υπολογιστή.

ΠΑΡΑΔΕΙΓΜΑ ΡΥ34, Γεωμετρική καταλληλότητα και επιλογή

```
class Box:
    def __init__(self, code, length, width, height):
        self.code = code
        self.length = length
        self.width = width
        self.height = height

    def volume(self):
        return self.length * self.width * self.height

    def fits(self, product, clearance=1):
        # Ένα cm σε ΚΑΘΕ πλευρά σημαίνει δύο cm ανά διάσταση.
        length, width, height = product
        length += 2 * clearance
        width += 2 * clearance
        height += 2 * clearance
        normal = length <= self.length and width <= self.width
        rotated = width <= self.length and length <= self.width
        return height <= self.height and (normal or rotated)

def best_box(boxes, product):
    """Επιστρέφει κατάλληλο κουτί ή None, χωρίς αλλαγή του καταλόγου."""
    candidates = [box for box in boxes if box.fits(product)]
    if not candidates:
        return None
    return min(candidates, key=lambda box: (box.volume(), box.code))

boxes = [Box(20, 35, 25, 8), Box(10, 32, 22, 7)]
best = best_box(boxes, (30, 20, 5))
if best is None:
    print("Δεν βρέθηκε κατάλληλο κουτί.")
else:
    print(best.code, best.volume())
```

Έξοδος της συγκεκριμένης δοκιμής

10 4928

Το διδακτικό παράδειγμα χρησιμοποιεί σταθερά ελεγμένα θετικά δεδομένα. Η κανονική καταχώριση του Project προσθέτει ελέγχους στον κατασκευαστή. Το κριτήριο είναι πλειάδα: πρώτα συγκρίνεται όγκος και σε ισότητα κωδικός.

Η περιστροφή που επιτρέπεται εδώ είναι μόνο οριζόντια: ανταλλάσσονται μήκος και πλάτος και το ύψος μένει ίδιο. Αν επιτρέπονταν όλες οι ανατροπές, θα χρειαζόταν άλλος κανόνας. Ένα πρόγραμμα δεν επιλέγει μόνο του διαφορετική φυσική ερμηνεία για να βρει περισσότερες λύσεις.

Δεδομένα	Συμπέρασμα
Προϊόν 30×20×5 με περιθώριο 1	Χρειάζεται 32×22×7.
Κουτί 32×22×7	Χωρά ακριβώς.
Κουτί 22×32×7	Χωρά με οριζόντια περιστροφή.
Κουτί 100×10×10	Ο όγκος δεν αρκεί: δεν χωρά στη βάση.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί η min δεν καλείται πριν ελέγξουμε αν υπάρχουν candidates;

Απάντηση και αιτιολόγηση, Η min κενής λίστας προκαλεί σφάλμα. Η απουσία κατάλληλου κουτιού είναι προβλεπόμενο αποτέλεσμα, γι' αυτό επιστρέφουμε None και εμφανίζουμε κατανοητό μήνυμα.

Τεκμηρίωση κεφαλαίου: [P7], [P4].

16 | Κατάσταση εφαρμογής και αμετάβλητοι κανόνες

Στόχος, Να προστατεύετε τους κανόνες ενός αντικειμένου και να ελέγχετε τον κατάλογο πριν από κάθε αλλαγή.

Η κατάσταση είναι τα δεδομένα που περιγράφουν ένα αντικείμενο ή μια συλλογή. Στον κατάλογο SmartPhones βασικός κανόνας είναι ότι η τιμή και η RAM παραμένουν θετικές. Ένας setter ελέγχει τον τοπικό κανόνα, ενώ οι συναρτήσεις του καταλόγου εφαρμόζουν φίλτρα χωρίς να αλλοιώνουν τα αρχικά αντικείμενα.

ΠΑΡΑΔΕΙΓΜΑ PY35, Έλεγχος τιμής και φίλτρο καταλόγου

```
class SmartPhone:
    def __init__(self, model, price, ram):
        self.model = model
        self.price = price
        self.ram = ram
    @property
    def price(self):
        return self.__price
    @price.setter
    def price(self, value):
        if value <= 0:
            raise ValueError("Η τιμή πρέπει να είναι θετική.")
        self.__price = float(value)
    def within_budget(phones, budget, minimum_ram):
        return [p for p in phones if p.price <= budget and p.ram >= minimum_ram]
phones = [SmartPhone("A", 500, 8), SmartPhone("B", 700, 12)]
print([p.model for p in within_budget(phones, 600, 8)])
try:
    phones[0].price = -10
except ValueError as error:
```

```
print(error)
print(phones[0].price)
```

Έξοδος της συγκεκριμένης δοκιμής

```
['A']
Η τιμή πρέπει να είναι θετική.
500.0
```

Η αποτυχημένη αλλαγή δεν καταστρέφει την προηγούμενη έγκυρη τιμή. Η `within_budget()` επιστρέφει νέα λίστα αναφορών και δεν αλλάζει τον αρχικό κατάλογο.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ποιος κανόνας πρέπει να ισχύει πάντα για την `price`;

Απάντηση και αιτιολόγηση, Η `price` πρέπει να παραμένει θετική. Δοκιμάζουμε τη δημιουργία και μια μεταγενέστερη άκυρη ανάθεση για να επιβεβαιώσουμε ότι ο κανόνας δεν παραβιάζεται.

Τεκμηρίωση κεφαλαίου: [P7], [P4].

17 | Αναζήτηση κατάλληλης λύσης και προσανατολισμοί

Στόχος, Να εξετάζετε συστηματικά ένα μικρό σύνολο υποψήφιων λύσεων και να επιλέγετε εκείνη που ικανοποιεί τους περιορισμούς.

Στο Θέμα Δ το προϊόν μπορεί να περιστραφεί. Οι τρεις διαστάσεις έχουν έως έξι διαφορετικές μεταθέσεις. Αντί να γράψουμε έξι χειροκίνητους ελέγχους, παράγουμε τους προσανατολισμούς και ελέγχουμε τον ίδιο κανόνα για όλους.

ΠΑΡΑΔΕΙΓΜΑ PY36, Έλεγχος όλων των προσανατολισμών προϊόντος

```
from itertools import permutations
def fits(box, product, clearance=1):
    required = tuple(value + 2 * clearance for value in product)
    for orientation in set(permutations(required)):
        if all(need <= have for need, have in zip(orientation, box)):
            return True
    return False
print(fits((32, 22, 7), (20, 30, 5)))
print(fits((30, 20, 5), (30, 20, 5)))
```

Έξοδος της συγκεκριμένης δοκιμής

```
True
False
```

Η πρώτη δοκιμή πετυχαίνει επειδή ένας προσανατολισμός του προϊόντος, μαζί με το περιθώριο, χωρά στο κουτί. Η δεύτερη αποτυγχάνει επειδή το περιθώριο αυξάνει κάθε απαιτούμενη διάσταση κατά 2 cm.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί χρησιμοποιούμε `set(permutations(...))`;

Απάντηση και αιτιολόγηση, Αν δύο διαστάσεις είναι ίδιες, οι `permutations` μπορεί να παράγουν ίδιες μεταθέσεις. Το `set` αφαιρεί τα διπλότυπα χωρίς να αλλάζει τη λογική του ελέγχου.

Τεκμηρίωση κεφαλαίου: [P12], [P4].

18 | Αναδρομή και αναζήτηση με επιστροφή

ΠΡΟΑΙΡΕΤΙΚΗ ΕΜΒΑΘΥΝΣΗ

Στόχος, Να καταλαβαίνετε μια συνάρτηση που καλεί τον εαυτό της και πώς αναιρείται μια δοκιμαστική επιλογή.

Αναδρομή σημαίνει ότι μια συνάρτηση λύνει ένα πρόβλημα καλώντας τον εαυτό της για μικρότερο πρόβλημα. Χρειάζεται οπωσδήποτε περίπτωση βάσης, στην οποία δεν γίνεται νέα κλήση, και

πρόοδος προς αυτήν. Διαφορετικά οι κλήσεις συνεχίζονται μέχρι να εξαντληθεί το όριο αναδρομής.

ΠΑΡΑΔΕΙΓΜΑ PY38, Παραγοντικό με γνωστή βάση

```
def factorial(n):
    """Δέχεται μικρό μη αρνητικό ακέραιο και επιστρέφει n!."""
    if type(n) is not int or n < 0:
        raise ValueError("Χρειάζεται μη αρνητικός ακέραιος.")
    if n == 0:
        return 1
    return n * factorial(n - 1)
print(factorial(4))
```

Έξοδος της συγκεκριμένης δοκιμής

24

Η αλυσίδα είναι $4 \times \text{factorial}(3)$, έπειτα $3 \times \text{factorial}(2)$, έως $\text{factorial}(0)=1$. Τα αποτελέσματα επιστρέφουν προς τα πίσω: 1, 1, 2, 6, 24. Δεν αυξάνουμε αυθαίρετα το όριο αναδρομής για να κρύψουμε κακό σχεδιασμό.

Backtracking: δοκιμάζω, συνεχίζω, αναιρώ

Στην αναζήτηση με επιστροφή κάνουμε μια επιλογή και εξετάζουμε αν οδηγεί σε λύση. Αν όχι, επιστρέφουμε στην προηγούμενη κατάσταση και δοκιμάζουμε άλλη. Στο επόμενο μικρό παράδειγμα αναζητούμε υποσύνολο αριθμών με συγκεκριμένο άθροισμα. Κάθε αριθμός μπορεί να χρησιμοποιηθεί το πολύ μία φορά, επειδή ο δείκτης προχωρά.

ΠΑΡΑΔΕΙΓΜΑ PY39, Υποσύνολο με συγκεκριμένο άθροισμα

```
def subset_sum(values, target):
    """Επιστρέφει μία επιλογή ή None. Δεν αλλάζει την αρχική λίστα."""
    # Η εσωτερική αναζήτηση δοκιμάζει μια επιλογή και επιστρέφει πίσω
    # όταν αυτή δεν οδηγεί στον στόχο.
    def search(index, selected, total):
        if total == target:
            return selected.copy()
        if index == len(values):
            return None
        selected.append(values[index])
        found = search(index + 1, selected, total + values[index])
        selected.pop() # Αναιρούμε την επιλογή για την άλλη διαδρομή.
        if found is not None:
            return found
        return search(index + 1, selected, total)
    return search(0, [], 0)
print(subset_sum([3, 5, 8], 11))
print(subset_sum([3, 5, 8], 1))
```

Έξοδος της συγκεκριμένης δοκιμής

[3, 8]
None

Η αντιγραφή της επιτυχημένης λίστας είναι απαραίτητη, επειδή η προσωρινή selected αλλάζει όταν επιστρέφουν οι κλήσεις. Δεν απορρίπτουμε $\text{total} > \text{target}$: ο γενικός κώδικας μπορεί να δεχθεί και αρνητικές τιμές.

Η ίδια ιδέα μπορεί να χρησιμοποιηθεί γενικά όταν εξετάζουμε διαδοχικές επιλογές και χρειάζεται να αναιρούμε μία δοκιμή που δεν οδηγεί σε αποδεκτό αποτέλεσμα. Η πλήρης αναζήτηση μπορεί να εξετάσει πολλές επιλογές, γι' αυτό χρησιμοποιείται μόνο όταν το μέγεθος του προβλήματος το επιτρέπει.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί το `found is not None` είναι καλύτερο από το `if found` στο παράδειγμα;

Απάντηση και αιτιολόγηση, Η κενή λίστα είναι έγκυρη λύση για στόχο 0. Είναι όμως ψευδής σε λογικό έλεγχο. Το `None` δηλώνει αποτυχία και πρέπει να ελέγχεται ρητά.

Τεκμηρίωση κεφαλαίου: [P3].

19 | Δοκιμές, σφάλματα και τεκμηρίωση

Στόχος, Να ελέγχετε σωστό αποτέλεσμα, προβλεπόμενη αποτυχία και αμετάβλητα δεδομένα.

Μια δοκιμή έχει δεδομένα εισόδου, αναμενόμενο αποτέλεσμα και τρόπο σύγκρισης. Δεν είναι δοκιμή να εκτελέσουμε κάτι και να θεωρήσουμε σωστό ό,τι εμφανιστεί. Πριν γράψουμε τη λειτουργία, υπολογίζουμε ένα μικρό παράδειγμα στο χαρτί και επιλέγουμε οριακές περιπτώσεις.

Σφάλμα	Συνήθης αιτία	Τι ελέγχουμε
<code>SyntaxError</code> / <code>IndentationError</code>	Λάθος σύνταξη ή εσοχές.	Προηγούμενη γραμμή, άνω-κάτω τελεία, στοίχιση.
<code>NameError</code>	Όνομα χωρίς τιμή ή λάθος γραφή.	Εμβέλεια και σειρά εκτέλεσης.
<code>TypeError</code>	Λειτουργία σε λάθος τύπο.	Είσοδος και επιστροφή συναρτήσεων.
<code>ValueError</code>	Αποδεκτός τύπος αλλά μη αποδεκτή τιμή.	Μετατροπή και όρια.
<code>IndexError</code> / <code>KeyError</code>	Θέση ή κλειδί που δεν υπάρχει.	Όρια λίστας και πεδία λεξικού.
<code>OSError</code>	Αποτυχία πρόσβασης αρχείου.	Διαδρομή, όνομα και δικαιώματα.

ΠΑΡΑΔΕΙΓΜΑ PY40, Έλεγχος επιτυχίας και μη μεταβολής μετά από λάθος

```
# Ελέγχουμε όχι μόνο το μήνυμα λάθους αλλά και ότι η αποτυχημένη ενέργεια
# διατήρησε τα δεδομένα αμετάβλητα.
def correct_grade(grades, number, value):
    if not 1 <= number <= len(grades):
        raise ValueError("Μη έγκυρος αριθμός φοιτητή.")
    if not 0 <= value <= 10:
        raise ValueError("Μη έγκυρος βαθμός.")
    grades[number - 1] = value

grades = [5, 5, 5]
correct_grade(grades, 1, 8)
assert grades == [8, 5, 5]
before = grades.copy()
try:
    correct_grade(grades, 1, 11)
except ValueError:
    pass
assert grades == before
print("Επιτυχία και προστασία κατάστασης ελέγχθηκαν.")
```

Έξοδος της συγκεκριμένης δοκιμής

Επιτυχία και προστασία κατάστασης ελέγχθηκαν.

Το παράδειγμα θεωρεί ήδη αριθμητικούς τύπους και ελέγχει όρια. Τα `assert` είναι εσωτερικές προσδοκίες δοκιμών. Δεν αντικαθιστούν τους ελέγχους εισόδου, γιατί μπορούν να απενεργοποιηθούν σε βελτιστοποιημένη εκτέλεση.

ΠΑΡΑΔΕΙΓΜΑ PY41, Δοκιμές με unittest

```
# Κάθε μέθοδος test περιγράφει μία προσδοκία. Ο runner τις εκτελεί
# ανεξάρτητα και συγκεντρώνει τα αποτελέσματα.
import unittest

def average(values):
    if not values:
        raise ValueError("Κενή λίστα.")
    return sum(values) / len(values)

class AverageTests(unittest.TestCase):
    def test_regular_case(self):
        self.assertEqual(average([6, 8, 10]), 8)

    def test_empty_input(self):
        with self.assertRaises(ValueError):
            average([])

suite = unittest.defaultTestLoader.loadTestsFromTestCase(AverageTests)
result = unittest.TestResult()
suite.run(result)
print(result.testsRun, result.wasSuccessful())
```

Έξοδος της συγκεκριμένης δοκιμής

```
2 True
```

Οι μέθοδοι με πρόθεμα test_ αποτελούν ξεχωριστές δοκιμές. Η assertAlmostEqual είναι κατάλληλη για προσεγγιστικό αριθμητικό αποτέλεσμα, ενώ η assertRaises ελέγχει αναμενόμενη εξαίρεση.

Ένα καλό σχόλιο εξηγεί τον σκοπό και τον περιορισμό: «αντιγράφουμε κάθε λεξικό ώστε το σενάριο να μην αλλάξει τις αρχικές ώρες». Δεν επαναλαμβάνει απλώς «κάνει copy». Η docstring περιγράφει τι δέχεται μια συνάρτηση, τι επιστρέφει και τι μεταβάλλει. Ένα παλιό λανθασμένο σχόλιο είναι χειρότερο από μια ειλικρινή σύντομη περιγραφή.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ποιος έλεγχος λείπει αν επιβεβαιώνουμε μόνο ότι μια αποτυχημένη επιστροφή συσκευής εμφανίζει ValueError;

Απάντηση και αιτιολόγηση, Πρέπει να ελέγξουμε ότι η προηγούμενη έγκυρη τιμή του αντικειμένου παραμένει αποθηκευμένη. Η εξαίρεση από μόνη της δεν αποδεικνύει ότι δεν έγινε μερική αλλαγή.

Τεκμηρίωση κεφαλαίου: [P6], [P14], [P15].

20 | Ολοκληρωμένο παράδειγμα: αναφορά βαθμολογίας

Στόχος, Να συνδέετε έλεγχο εισόδου, λίστα εγγραφών, στατιστική ανάλυση και καθαρή παρουσίαση.

Θα καταχωρίσουμε μικρό αριθμό φοιτητών, θα υπολογίσουμε μέσο όρο και θα εμφανίσουμε όσους έχουν βαθμό αυστηρά πάνω από αυτόν. Η είσοδος, ο υπολογισμός και η παρουσίαση παραμένουν χωριστές ευθύνες.

ΠΑΡΑΔΕΙΓΜΑ PY42, Πλήρες μικρό πρόγραμμα βαθμολογίας

```
def read_grade(message):
    while True:
        try:
            value = float(input(message).replace(", ", "."))
            if 0 <= value <= 10:
                return value
        except ValueError:
            pass
    print("Δώστε βαθμό από 0 έως 10.")

def analyze(students):
    average = sum(s["grade"] for s in students) / len(students)
    above = [s for s in students if s["grade"] > average]
```

```
return average, above
def main():
    students = []
    for i in range(1, 4):
        name = input(f"Όνομα {i}: ").strip()
        grade = read_grade("Βαθμός: ")
        students.append({"name": name, "grade": grade})
    average, above = analyze(students)
    print(f"Μέσος όρος: {average:.2f}")
    print("Πάνω από τον μέσο όρο:")
    for student in above:
        print(student["name"], student["grade"])
if __name__ == "__main__":
    main()
```

Ενδεικτική είσοδος: Άννα 9, Νίκος 7, Μαρία 4.

Έξοδος της συγκεκριμένης δοκιμής

```
Μέσος όρος: 6.67
Πάνω από τον μέσο όρο:
Άννα 9.0
Νίκος 7.0
```

Γιατί γράφτηκε με αυτή τη σειρά

Η `read_grade()` εγγυάται έγκυρο βαθμό πριν γίνει καταχώριση. Η `analyze()` δεν εκτυπώνει, αλλά επιστρέφει αποτελέσματα. Το `main()` συνδέει τα βήματα και παρουσιάζει την τελική αναφορά.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ποιες δοκιμές προσθέτετε στο πλήρες παράδειγμα;

Απάντηση και αιτιολόγηση, Βαθμούς 0 και 10, κείμενο αντί αριθμού, τρεις ίδιους βαθμούς, έναν πολύ υψηλό και δύο χαμηλούς, καθώς και δεκαδικές τιμές στα όρια 5.0, 6.5 και 8.5.

Τεκμηρίωση κεφαλαίου: [P5], [P6], [P10].

21 | Χάρτης μελέτης για τα τέσσερα θέματα

Στόχος, Να συνδέετε τις έννοιες με τα τέσσερα θέματα του Project και να οργανώνετε προσωπικό έλεγχο προόδου.

Θέμα Α, επιστημονική αριθμομηχανή: είσοδος, συναρτήσεις, `math`, μενού, έλεγχος ειδικών περιπτώσεων και ιστορικό.

Θέμα Β, βαθμολογία εργαστηρίου: λίστες, λεξικά, στατιστικά, φίλτρα, ταξινόμηση και αναζήτηση.

Θέμα Γ, `SmartPhones`: κλάσεις, `self`, `properties`, έλεγχος ιδιοτήτων, λίστες αντικειμένων και φίλτρα.

Θέμα Δ, συσκευασία: κλάση `Box`, γεωμετρικοί υπολογισμοί, προσανατολισμοί, αναζήτηση κατάλληλου κουτιού και επιλογή καλύτερης λύσης.

Μικρό γλωσσάρι

Συνάρτηση: επαναχρησιμοποιήσιμο τμήμα κώδικα με σαφή είσοδο και αποτέλεσμα. Λίστα: διατεταγμένη συλλογή. Λεξικό: εγγραφή με ονομασμένα κλειδιά. Κλάση: πρότυπο δημιουργίας αντικειμένων. Property: ελεγχόμενη πρόσβαση σε ιδιότητα.

Διαδρομή από την ιδέα στον έλεγχο

1. Γράφω με απλά λόγια τι παίρνει και τι δίνει το πρόγραμμα, μαζί με μονάδες και όρια.
2. Χωρίζω τις εργασίες σε μικρές συναρτήσεις και επιλέγω λίστα, λεξικό ή κλάση με βάση το νόημα των δεδομένων.

3. Κατασκευάζω ένα μικρό έγκυρο παράδειγμα και ένα παράδειγμα αποτυχίας που μπορώ να ελέγξω με το χέρι.

4. Προσθέτω είσοδο και μηνύματα. Δεν κρύβω λογικά λάθη μέσα σε γενικά except.

5. Δοκιμάζω καθαρή εκτέλεση, όρια, ισοβαθμίες και δεδομένα που πρέπει να παραμένουν αμετάβλητα μετά από αποτυχία.

Δοκιμάστε πριν διαβάσετε την απάντηση, Πώς καταλαβαίνω ότι έμαθα μια έννοια και δεν αντέγραψα απλώς κώδικα;

Απάντηση και αιτιολόγηση, Μπορώ να προβλέψω την έξοδο για νέα δεδομένα, να εξηγήσω έναν έλεγχο και να γράψω δική μου δοκιμή που θα εντόπιζε ένα συγκεκριμένο λάθος.

Πηγές και συμπληρωματική μελέτη

Επίσημη τεκμηρίωση Python για τις λειτουργίες που μελετήσαμε. Οι αριθμητικοί κανόνες και τα εκπαιδευτικά σενάρια ορίζονται στις αντίστοιχες ενότητες.

[P1] IDLE και Shell

[P2] Πρώτες εντολές

[P3] Συνθήκες και συναρτήσεις

[P4] Δομές δεδομένων

[P5] Είσοδος, έξοδος και αρχεία

[P6] Σφάλματα και εξαιρέσεις

[P7] Κλάσεις και εμβέλεια

[P8] Αρθρώματα και εισαγωγές

[P9] Δεκαδική αριθμητική

[P10] Μαθηματικές συναρτήσεις

[P11] Τυχαίοι αριθμοί

[P12] Ουρά deque

[P13] JSON

[P14] Δοκιμές unittest

[P15] Ενσωματωμένες συναρτήσεις

[P16] Ουρά προτεραιότητας heapq

[P17] Διαδρομές αρχείων

[P18] Αρχεία CSV

[P19] Τύποι και λογικοί έλεγχοι