



Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ

Τεχνικές Προγραμματισμού Υπολογιστών

Εργαστηριακό μάθημα • Α' εξάμηνο

JavaScript 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.

JavaScript στην κονσόλα και στα πλαίσια διαλόγου.

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 33 αριθμημένα παραδείγματα. Το κάθε ένα συνοδεύεται από αρχείο στον φάκελο Examples/JavaScript. 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	Από το πρόβλημα στο πρώτο πρόγραμμα	Βασικός κορμός
2	Τιμές, τύποι και μεταβλητές	Βασικός κορμός
3	Πράξεις, μονάδες και αριθμητικά όρια	Βασικός κορμός
4	Κονσόλα και πλαίσια διαλόγου	Βασικός κορμός
5	Συνθήκες και λογικές αποφάσεις	Βασικός κορμός
6	Επαναλήψεις χωρίς παγίδες	Βασικός κορμός
7	Συναρτήσεις, παράμετροι και επιστροφή αποτελέσματος	Βασικός κορμός
8	Πίνακες: πολλές τιμές με σωστή σειρά	Βασικός κορμός
9	Αλγόριθμοι πάνω σε πίνακες	Βασικός κορμός
10	Αντικείμενα, ιδιότητες και μέθοδοι	Βασικός κορμός
11	Έλεγχος εισόδου και διαχείριση σφαλμάτων	Βασικός κορμός
12	Μαθηματικά προβλήματα με σαφή βήματα	Βασικός κορμός
13	Τυχαιότητα και κατάσταση παιχνιδιού	Βασικός κορμός
14	Ένα πλήρες μενού μόνο στην κονσόλα	Βασικός κορμός
15	Εμβέλεια, αναφορές και κλεισίματα	Βασικός κορμός
16	Κλάσεις και ελεγχόμενες ιδιότητες	Εμβάθυνση

Κεφ.	Τι θα μελετήσετε	Πορεία
17	Κείμενα, Set και οργανωμένες συλλογές	Εμβάθυνση
18	Αποσφαλμάτωση και δοκιμές που αποδεικνύουν κάτι	Βασικός κορμός
19	Ολοκληρωμένο παράδειγμα: μικρή αναφορά βαθμολογίας	Βασικός κορμός
20	Από τη μελέτη στην υλοποίηση των θεμάτων	Βασικός κορμός

01 | Από το πρόβλημα στο πρώτο πρόγραμμα

Στόχος, Να ξεχωρίζετε το πρόγραμμα, τα δεδομένα και το περιβάλλον εκτέλεσης και να εκτελείτε ένα αρχείο JavaScript χωρίς ιστοσελίδα.

Προγραμματίζω σημαίνει ότι περιγράφω με ακρίβεια μια σειρά ενεργειών. Δεν αρκεί να γράψω «βρες τον μέσο όρο»: πρέπει να ορίσω ποιους αριθμούς θα χρησιμοποιήσω, πώς θα τους προσθέσω, με ποιο πλήθος θα διαιρέσω και τι θα γίνει όταν τα δεδομένα λείπουν. Αυτή η συγκεκριμένη σειρά βημάτων ονομάζεται αλγόριθμος.

Στάδιο	Ερώτηση που κάνουμε	Παράδειγμα
Είσοδος	Τι γνωρίζουμε;	Τρεις χρόνοι: 12, 15, 9 λεπτά.
Επεξεργασία	Ποιες ενέργειες χρειάζονται;	Πρόσθεση και διαίρεση με το 3.
Έξοδος	Τι θέλουμε να ανακοινώσουμε;	Μέσος χρόνος: 12 λεπτά.

Η JavaScript είναι η γλώσσα με την οποία γράφουμε τις εντολές. Ο φυλλομετρητής είναι ένα περιβάλλον που μπορεί να τις εκτελέσει. Στις σημειώσεις χρησιμοποιούμε την κονσόλα του για τα αποτελέσματα και τα ενσωματωμένα πλαίσια διαλόγου για είσοδο. Δεν κατασκευάζουμε ιστοσελίδα και δεν χρειάζονται HTML, CSS, κουμπιά ή κώδικας DOM.

Πρώτη εκτέλεση σε Chrome DevTools

1. Ανοίξτε μια κενή καρτέλα και γράψτε `about:blank` στη γραμμή διεύθυνσης. Η κενή επιφάνεια δεν είναι ο χώρος των αποτελεσμάτων.
2. Ανοίξτε τα εργαλεία προγραμματιστή με F12. Μέσα στα εργαλεία πατήστε `Ctrl+Shift+P`, αναζητήστε `Show Snippets` και επιλέξτε το.
3. Πατήστε `New snippet`, δώστε ένα όνομα και γράψτε ολόκληρο το μικρό πρόγραμμα που ακολουθεί. Με `Ctrl+Enter` το εκτελείτε.
4. Διαβάστε την καρτέλα `Console`. Αλλάξτε έναν χρόνο, εκτελέστε ξανά και εξηγήστε γιατί άλλαξε ο μέσος όρος.
5. Φυλάξτε και ένα αντίγραφο σε αρχείο `.js`. Η αποθήκευση ενός snippet δεν ενημερώνει αυτόματα το αρχείο του δίσκου.

ΠΑΡΑΔΕΙΓΜΑ JS01, Το πρώτο αποτέλεσμα στην κονσόλα

```
// Τα λεπτά αποθηκεύονται ως αριθμοί, ώστε να γίνουν πράξεις.
const first = 12;
const second = 15;
const third = 9;
const average = (first + second + third) / 3;
console.log("Μέσος χρόνος:", average, "λεπτά");
```

Έξοδος της συγκεκριμένης δοκιμής

Μέσος χρόνος: 12 λεπτά

Η ανάθεση αποθηκεύει αποτέλεσμα· η `console.log` το εμφανίζει. Οι παρενθέσεις εξασφαλίζουν ότι πρώτα προστίθενται και οι τρεις χρόνοι.

Επαναλαμβανόμενη εκτέλεση, Στα μικρά παραδείγματα μπορείτε να περικλείετε όλο τον κώδικα σε ένα ζεύγος `{ και }`. Έτσι οι δηλώσεις `let/const` μένουν μέσα σε εκείνο το μπλοκ. Τα συνοδευτικά αρχεία παραδειγμάτων έχουν ήδη αυτό το περίβλημα. Εκτελείτε ένα παράδειγμα κάθε φορά και μην επικολλάτε άγνωστο κώδικα παρακάμπτοντας προειδοποιήσεις ασφαλείας.

Ορισμένες κονσόλες εμφανίζουν επιπλέον `undefined` μετά την εκτέλεση. Αυτό μπορεί να είναι η επιστρεφόμενη τιμή της τελευταίας εντολής και όχι μήνυμα αποτυχίας. Ξεχωρίζετε τις γραμμές που τυπώνει το πρόγραμμα από τις πληροφορίες του περιβάλλοντος.

Δοκιμάστε πριν διαβάσετε την απάντηση, Αν οι χρόνοι είναι 10, 20 και 30, ποια είναι η έξοδος; Γιατί το `first + second + third / 3` είναι διαφορετικός υπολογισμός;

Απάντηση και αιτιολόγηση, Ο σωστός μέσος είναι 20. Χωρίς τις παρενθέσεις διαιρείται μόνο το 30: ο υπολογισμός γίνεται $10 + 20 + 10 = 40$.

Τεκμηρίωση κεφαλαίου: [1], [6].

02 | Τιμές, τύποι και μεταβλητές

Στόχος, Να επιλέγετε περιγραφικά ονόματα και να καταλαβαίνετε τι ακριβώς αλλάζει σε κάθε ανάθεση.

Μια μεταβλητή συνδέει ένα όνομα με μια τιμή. Το όνομα δεν είναι η τιμή: μπορούμε να αλλάξουμε τα λίτρα που αποθηκεύει το `water` χωρίς να αλλάξουμε το όνομα. Το `let` χρησιμοποιείται όταν θα γίνει νέα ανάθεση. Το `const` δηλώνει ότι δεν θα δώσουμε άλλη τιμή στο ίδιο όνομα μετά τη δημιουργία του.

Τύπος ή ειδική τιμή	Παράδειγμα	Σημασία
number	18, 2.5, -3	Αριθμητικός υπολογισμός.
string	"18", "Αννα"	Κείμενο, ακόμη και όταν περιέχει ψηφία.
boolean	true, false	Αληθές ή ψευδές.
null	let selected = null;	Δηλώνουμε σκόπιμα ότι δεν υπάρχει επιλογή.
undefined	let result;	Δεν έχει δοθεί ακόμη τιμή.

ΠΑΡΑΔΕΙΓΜΑ JS02, Η ανάθεση διαβάζει πρώτα τη δεξιά πλευρά

```
let water = 40;
// Η νέα ποσότητα υπολογίζεται από την παλιά και μετά αποθηκεύεται.
water = water - 7;
water = water + 2;
console.log(water);
console.log(typeof water);
console.log(typeof "35");
```

Έξοδος της συγκεκριμένης δοκιμής

```
35
number
string
```

Το = δεν είναι μαθηματική ισότητα. Η εντολή `water = water - 7` σημαίνει «διάβασε την παλιά τιμή, αφάιρεσε 7 και αποθήκευσε το αποτέλεσμα».

Εντολή	Τιμή πριν	Τιμή μετά
let water = 40	Δεν υπάρχει ακόμη	40
water = water - 7	40	33
water = water + 2	33	35

Ονόματα όπως quantity, radius και studentName εξηγούν τη χρήση τους. Η JavaScript ξεχωρίζει κεφαλαία από πεζά: score και Score είναι διαφορετικά ονόματα. Τα ονόματα δεν αρχίζουν με ψηφίο και δεν πρέπει να είναι λέξεις της γλώσσας, όπως if. Στα παραδείγματα τα ονόματα είναι λατινικά για εύκολη πληκτρολόγηση, ενώ τα σχόλια και τα μηνύματα είναι ελληνικά.

const δεν σημαίνει αμετάβλητα περιεχόμενα, Ένας πίνακας δηλωμένος const μπορεί να αποκτήσει στοιχεία με push. Αυτό που απαγορεύεται είναι να αναθέσουμε στο ίδιο όνομα έναν διαφορετικό πίνακα. Η διάκριση θα φανεί αναλυτικά στο κεφάλαιο των πινάκων.

Δοκιμάστε πριν διαβάσετε την απάντηση, Τι δίνει το let a = 4; let b = a; a = 9; όταν εμφανίσουμε b;

Απάντηση και αιτιολόγηση, Δίνει 4. Το b έλαβε την αριθμητική τιμή που είχε τότε το a. Δεν δημιουργήθηκε σύνδεση που να ενημερώνει αυτόματα το b.

Τεκμηρίωση κεφαλαίου: [2], [16].

03 | Πράξεις, μονάδες και αριθμητικά όρια

Στόχος, Να μεταφράζετε έναν τύπο σε σωστή έκφραση και να ξεχωρίζετε τον υπολογισμό από τη μορφοποίηση.

Πριν πληκτρολογήσετε έναν τύπο, γράψτε τις μονάδες των δεδομένων. Αν μια διάσταση είναι σε μέτρα και μια άλλη σε εκατοστά, η απευθείας αντικατάσταση στον ίδιο τύπο δίνει λανθασμένο αποτέλεσμα, ακόμη κι αν ο κώδικας εκτελείται κανονικά. Οι παρενθέσεις κάνουν τη σειρά πράξεων σαφή και για τον υπολογιστή και για τον αναγνώστη.

Γραφή	Λειτουργία	Παράδειγμα αποτελέσματος
+ - * /	Άθροισμα, διαφορά, γινόμενο, πηλίκο	8 / 2 → 4
**	Ύψωση σε δύναμη	3 ** 2 → 9
%	Υπόλοιπο διαίρεσης	17 % 5 → 2
Math.floor(x)	Μικρότερος ή ίσος ακέραιος	Math.floor(3.9) → 3
Math.abs(x)	Απόλυτη τιμή	Math.abs(-4) → 4
toFixed(2)	Κείμενο με δύο δεκαδικά	(3.5).toFixed(2) → "3.50"

ΠΑΡΑΔΕΙΓΜΑ JS03, Κουτιά πλήρους χωρητικότητας

```
const items = 23;
const capacity = 5;
// Μετράμε μόνο τα κουτιά που γεμίζουν εντελώς.
const fullBoxes = Math.floor(items / capacity);
const remaining = items % capacity;
console.log("Γεμάτα κουτιά:", fullBoxes);
console.log("Αντικείμενα που απομένουν:", remaining);
```

Έξοδος της συγκεκριμένης δοκιμής

Γεμάτα κουτιά: 4
Αντικείμενα που απομένουν: 3

Το 23 / 5 δίνει 4.6, αλλά δεν υπάρχουν 4.6 γεμάτα κουτιά. Το πηλίκιο και το υπόλοιπο απαντούν σε διαφορετικές ερωτήσεις.

ΠΑΡΑΔΕΙΓΜΑ JS04, Η εμφάνιση δεν αλλάζει τον αρχικό αριθμό

```
const average = 20 / 3;
const displayed = average.toFixed(2);
console.log(displayed);
console.log(typeof displayed);
// Συγκρίνουμε με τον πραγματικό αριθμό, όχι με την εκτυπωμένη μορφή.
console.log(6.668 > average);
console.log(0.1 + 0.2 === 0.3);
```

Έξοδος της συγκεκριμένης δοκιμής

```
6.67
string
true
false
```

Η toFixed επιστρέφει κείμενο. Το τελευταίο false δείχνει ότι ορισμένα δεκαδικά δεν αποθηκεύονται ακριβώς σε δυαδική μορφή. Δεν σημαίνει ότι η πρόσθεση «δεν λειτουργεί».

Η Number.isFinite(value) απορρίπτει NaN και Infinity. Η διαίρεση με μηδέν μπορεί να δώσει Infinity και η Math.sqrt(-1) δίνει NaN. Γι' αυτό ελέγχουμε πρώτα τον παρονομαστή ή το πεδίο ορισμού και έπειτα το αποτέλεσμα. Για μεγάλους ακραίους υπάρχει και η Number.isSafeInteger: η αποθήκευση ενός πολύ μεγάλου πλήθους μπορεί να μην είναι ακριβής.

Ανοχή με νόημα, Δεν υπάρχει ένα μαγικό μικρό νούμερο που διορθώνει κάθε αριθμητικό πρόβλημα. Η ανοχή πρέπει να ορίζεται από την άσκηση και την κλίμακα των δεδομένων. Για έναν έλεγχο ακρίβειας συγκρίνουμε Math.abs(actual - expected) με ένα μικρό επιτρεπτό σφάλμα, όχι δύο μορφοποιημένα κείμενα.

Δοκιμάστε πριν διαβάσετε την απάντηση, Πώς υπολογίζονται οι πλήρεις ημέρες για 1250 λίτρα και 400 λίτρα κατανάλωση την ημέρα; Τι αλλάζει αν η κατανάλωση είναι 0;

Απάντηση και αιτιολόγηση, Math.floor(1250 / 400) = 3. Με κατανάλωση 0 δεν κάνουμε διαίρεση· ανακοινώνουμε ότι δεν καταναλώνεται νερό. Δεν εμφανίζουμε Infinity ως αριθμό ημερών.

Τεκμηρίωση κεφαλαίου: [7], [15], [16].

04 | Κονσόλα και πλαίσια διαλόγου

Στόχος, Να παίρνετε είσοδο με prompt και να διαχειρίζεστε σωστά την Ακύρωση, το κενό και τις επιβεβαιώσεις.

Η prompt εμφανίζει μια ερώτηση και επιστρέφει κείμενο. Αν ο χρήστης γράψει 8, η πρώτη τιμή που παίρνουμε είναι "8". Η μετατροπή σε αριθμό είναι ξεχωριστό βήμα. Η Ακύρωση επιστρέφει null. Το κενό πεδίο επιστρέφει "" και δεν πρέπει να αντιμετωπίζεται σαν Ακύρωση.

Εντολή	Τι κάνει	Τι επιστρέφει
console.log(...)	Γράφει αποτελέσματα στην κονσόλα.	Δεν ζητά απάντηση.
prompt("Ερώτηση")	Ζητά κείμενο από τον χρήστη.	Κείμενο ή null.
alert("Μήνυμα")	Παρουσιάζει σύντομη ειδοποίηση.	Δεν επιστρέφει επιλογή χρήστη.
confirm("Να συνεχίσω;")	Ζητά επιβεβαίωση μιας ενέργειας.	true για OK, false για άρνηση.

ΠΑΡΑΔΕΙΓΜΑ JS05, Μία ερώτηση με πλήρεις ελέγχους

```
// Ελέγχουμε πρώτα την Ακύρωση και το κενό. Μόνο πραγματικό κείμενο
// εισόδου μετατρέπεται σε αριθμό.
const raw = prompt("Δώστε βαθμό από 0 έως 10:");
if (raw === null) {
  console.log("Η εισαγωγή ακυρώθηκε.");
} else if (raw.trim() === "") {
  console.log("Το κενό δεν είναι βαθμός.");
} else {
  const grade = Number(raw.trim().replace(",", "."));
  if (Number.isFinite(grade) && grade >= 0 && grade <= 10) {
    console.log("Βαθμός:", grade.toFixed(2));
  } else {
    console.log("Χρειάζεται έγκυρος αριθμός από 0 έως 10.");
  }
}
```

Δοκιμή εισόδου, με τη σειρά: 8,5.

Έξοδος της συγκεκριμένης δοκιμής

Βαθμός: 8.50

Το παράδειγμα κάνει μία μόνο προσπάθεια. Η επανάληψη της ερώτησης θα προστεθεί όταν μελετήσουμε τις while. Ο έλεγχος του null προηγείται της trim, γιατί το null δεν είναι κείμενο.

Η Number δέχεται και μορφές όπως "1e2". Ο παραπάνω μικρός έλεγχος δείχνει τη διαφορά κειμένου και αριθμού· δεν επιβάλλει ακόμη τον αυστηρό κανόνα «μόνο απλή δεκαδική γραφή». Στο κεφάλαιο 11 κατασκευάζουμε αναγνώστη που ελέγχει και τους χαρακτήρες. Άλλο πράγμα η δυνατότητα της γλώσσας και άλλο ο κανόνας που επιλέγει μια εκφώνηση.

ΠΑΡΑΔΕΙΓΜΑ JS06, Η άρνηση μιας ενέργειας δεν είναι έξοδος από την εφαρμογή

```
// Η απάντηση της confirm αφορά μόνο τη διαγραφή. Δεν τη χρησιμοποιούμε ως
// εντολή τερματισμού της εφαρμογής.
const accepted = confirm("Να διαγραφεί το πρόχειρο;");
if (accepted) {
  console.log("Η διαγραφή εγκρίθηκε.");
} else {
  console.log("Το πρόχειρο διατηρείται.");
}
console.log("Η εφαρμογή συνεχίζει.");
```

Απαντήσεις confirm: Ακύρωση / Όχι.

Έξοδος της συγκεκριμένης δοκιμής

Το πρόχειρο διατηρείται.
Η εφαρμογή συνεχίζει.

Στην confirm το false σημαίνει «όχι στην ερώτηση που τέθηκε». Δεν είναι υποχρεωτικά εντολή τερματισμού.

Σαφή μηνύματα. Οι αναλυτικές αναφορές μένουν στην κονσόλα. Ένα alert χρησιμοποιείται μόνο όταν χρειάζεται να προσέξει ο χρήστης ένα σύντομο μήνυμα. Αν θέλουμε να διατηρηθεί και ιστορικό, εμφανίζουμε το ίδιο μήνυμα και με console.log. Τα πλαίσια μπορεί να περιορίζονται από τον φυλλομετρητή σε ανενεργές καρτέλες· δουλεύουμε στην ενεργή καρτέλα της άσκησης.

Δοκιμάστε πριν διαβάσετε την απάντηση, Τι διαφορά έχουν οι είσοδοι null, "", "0";

Απάντηση και αιτιολόγηση, Η πρώτη είναι Ακύρωση. Η δεύτερη είναι κενό κείμενο. Η τρίτη είναι κείμενο που παριστάνει τον αριθμό μηδέν και μπορεί να είναι έγκυρο δεδομένο, π.χ. βαθμός 0.

Τεκμηρίωση κεφαλαίου: [J3], [J4], [J5], [J6].

05 | Συνθήκες και λογικές αποφάσεις

Στόχος, Να γράφετε ελέγχους με σαφή όρια και να αποφεύγετε διπλές ή αδύνατες περιπτώσεις.

Μια συνθήκη είναι μια ερώτηση που παίρνει απάντηση true ή false. Το if εκτελεί το σώμα του μόνο όταν η συνθήκη ισχύει. Το else περιγράφει την αντίθετη περίπτωση. Μια αλυσίδα if / else if / else επιλέγει μία διαδρομή, ενώ πολλά ανεξάρτητα if μπορούν να εκτελέσουν περισσότερες από μία ενέργειες.

ΠΑΡΑΔΕΙΓΜΑ JS07, Μία και μόνο κατηγορία για κάθε τιμή

```
const grade = 8.5;
let category;
// Οι περιπτώσεις εξετάζονται από το υψηλότερο όριο προς τα κάτω.
if (grade >= 8.5) {
  category = "Άριστα";
} else if (grade >= 6.5) {
  category = "Λίαν Καλώς";
} else if (grade >= 5) {
  category = "Καλώς";
} else {
  category = "Αποτυχία";
}
console.log(category);
```

Έξοδος της συγκεκριμένης δοκιμής

Άριστα

Όταν περάσει ο πρώτος έλεγχος, οι χαμηλότερες περιπτώσεις παραλείπονται. Με τέσσερα ανεξάρτητα if θα μπορούσαμε να μετρήσουμε τον ίδιο φοιτητή σε περισσότερες κατηγορίες.

Έκφραση	Ερμηνεία
value >= 0 && value <= 10	Και οι δύο περιορισμοί πρέπει να ισχύουν.
choice === 1 choice === 2	Αρκεί να ισχύει ένας από τους δύο.
!found	Δεν έχει βρεθεί αυτό που ψάχνουμε.
a === b	Ίση τιμή και ίδιος τύπος.
a !== b	Δεν είναι αυστηρά ίσα.

ΠΑΡΑΔΕΙΓΜΑ JS08, Σύγκριση αριθμού και κειμένου

```
// Η αυστηρή ισότητα συγκρίνει και τύπο. Μετατρέπουμε ρητά το κείμενο όταν
// επιδιώκουμε αριθμητική σύγκριση.
console.log(5 === "5");
console.log(5 === Number("5"));
const age = 18;
const hasTicket = true;
console.log(age >= 18 && hasTicket);
```

Έξοδος της συγκεκριμένης δοκιμής

false
true
true

Η ρητή μετατροπή κάνει την πρόθεση ορατή. Αποφεύγουμε να βασίζουμε την άσκηση στις αυτόματες μετατροπές της χαλαρής ισότητας ==.

Στην JavaScript το 0 <= grade <= 10 δεν εκφράζει σωστά διάστημα. Η πρώτη σύγκριση παράγει λογική τιμή και μετά γίνεται δεύτερη σύγκριση πάνω σε εκείνη. Γράφουμε grade >= 0 && grade <=

10. Επίσης, το if (raw) δεν αρκεί για έλεγχο αριθμού: το κείμενο "0" είναι μη κενό, άρα θεωρείται αληθές.

Πίνακας ορίων πριν από τον κώδικα

Βαθμός	Αναμενόμενη κατηγορία
4.99	Αποτυχία
5.00	Καλώς
6.50	Λίαν Καλώς
8.50	Άριστα
10.01	Μη έγκυρη είσοδος, όχι νέα κατηγορία

Δοκιμάστε πριν διαβάσετε την απάντηση, Πότε χρησιμοποιούμε δύο ανεξάρτητα if αντί για if/else;

Απάντηση και αιτιολόγηση, Όταν δύο γεγονότα μπορούν να συμβούν ταυτόχρονα. Για παράδειγμα, μια μέτρηση μπορεί να έχει και μεγάλη θερμοκρασία και μεγάλη υγρασία. Και οι δύο προειδοποιήσεις πρέπει τότε να εμφανιστούν.

Τεκμηρίωση κεφαλαίου: [7], [8].

06 | Επαναλήψεις χωρίς παγίδες

Στόχος, Να διαλέγετε for ή while και να παρακολουθείτε μετρητές, αθροίσματα και συνθήκες τερματισμού.

Η επανάληψη επιτρέπει να περιγράψουμε μια δουλειά μία φορά και να την εκτελέσουμε για πολλές τιμές. Η for με μετρητή ταιριάζει όταν γνωρίζουμε το πλήθος. Η for...of περνά από τα στοιχεία μιας συλλογής. Η while ταιριάζει όταν η συνέχεια εξαρτάται από μια συνθήκη, όπως «ο χρήστης δεν έδωσε ακόμη έγκυρη τιμή».

ΠΑΡΑΔΕΙΓΜΑ JS09, Μετρητής και άθροισμα δεν είναι το ίδιο

```
// Το step είναι ο μετρητής του βρόχου. Το total συγκεντρώνει όλες τις
// τιμές που έχουν ήδη εξεταστεί.
let total = 0;
for (let step = 1; step <= 4; step++) {
  total += step;
  console.log("Βήμα:", step, "Αθροισμα:", total);
}
```

Έξοδος της συγκεκριμένης δοκιμής

```
Βήμα: 1 Αθροισμα: 1
Βήμα: 2 Αθροισμα: 3
Βήμα: 3 Αθροισμα: 6
Βήμα: 4 Αθροισμα: 10
```

Το step απαντά «σε ποιο βήμα βρίσκομαι;». Το total απαντά «πόσο έχω συγκεντρώσει;». Η αύξηση step++ δεν πρέπει να αντικαταστήσει την total += step.

ΠΑΡΑΔΕΙΓΜΑ JS10, Περνάμε από δεδομένα και παραλείπουμε μια ειδική τιμή

```
const readings = [18, -1, 21, 20];
let total = 0;
let count = 0;
for (const reading of readings) {
  // Στο συγκεκριμένο δείγμα το -1 δηλώνει αποτυχημένη μέτρηση.
  if (reading === -1) continue;
  total += reading;
  count++;
}
```

```

}
if (count > 0) console.log("Μέσος:", total / count);
else console.log("Δεν υπάρχουν μετρήσεις.");

```

Έξοδος της συγκεκριμένης δοκιμής

Μέσος: 19.666666666666668

Η ειδική τιμή -1 είναι κανόνας αυτού του παραδείγματος. Δεν απορρίπτουμε γενικά κάθε αρνητική θερμοκρασία. Η διαίρεση χρησιμοποιεί το πλήθος των αποδεκτών μετρήσεων.

Τερματισμός και αλλαγή κατάστασης

Μια while πρέπει να έχει τρόπο να σταματήσει: να αλλάζει η μεταβλητή της συνθήκης, να εκτελείται break ή να ολοκληρώνεται η συνάρτηση με return. Το break σταματά μόνο την πλησιέστερη επανάληψη. Το continue παραλείπει τον υπόλοιπο τρέχοντα γύρο και προχωρά στον επόμενο. Στη do...while το σώμα εκτελείται τουλάχιστον μία φορά, επειδή ο έλεγχος γίνεται στο τέλος.

ΠΑΡΑΔΕΙΓΜΑ JS11, Αναζήτηση που σταματά μόλις βρεθεί στόχος

```

// Η θέση -1 δηλώνει αποτυχία αναζήτησης. Σταματάμε στην πρώτη εμφάνιση
// για να μην αντικαταστήσουμε την επιλογή.
const codes = [12, 17, 23, 31];
let position = -1;
for (let i = 0; i < codes.length; i++) {
  if (codes[i] === 23) {
    position = i;
    break; // Δεν χρειάζεται να εξετάσουμε τα επόμενα στοιχεία.
  }
}
console.log("Θέση πίνακα:", position);

```

Έξοδος της συγκεκριμένης δοκιμής

Θέση πίνακα: 2

Η τιμή -1 σημαίνει «δεν βρέθηκε» και δεν χρησιμοποιείται ως κανονική θέση. Η πρώτη θέση του πίνακα είναι 0.

Πριν εκτελέσετε έναν βρόχο, Γράψτε με το χέρι την πρώτη και την τελευταία τιμή του μετρητή. Ελέγξτε αν θέλετε < ή <=. Ένας βρόχος που δουλεύει για 5 στοιχεία μπορεί να αποτύχει για 0 ή 1. Μην χρησιμοποιείτε ατέρμονους βρόχους για δοκιμή στην κονσόλα· μπορούν να εμποδίσουν την απόκριση της καρτέλας.

Δοκιμάστε πριν διαβάσετε την απάντηση, Πόσες φορές εκτελείται for (let i = 0; i < 3; i++) και ποιες τιμές παίρνει το i;

Απάντηση και αιτιολόγηση, Τρεις φορές, με i ίσο με 0, 1 και 2. Η τιμή 3 ελέγχεται αλλά δεν εκτελεί το σώμα.

Τεκμηρίωση κεφαλαίου: [9].

07 | Συναρτήσεις, παράμετροι και επιστροφή αποτελέσματος

Στόχος, Να χωρίζετε ένα πρόγραμμα σε μικρές εργασίες και να ξεχωρίζετε τον υπολογισμό από την εμφάνιση.

Μια συνάρτηση είναι μια ονομασμένη ομάδα εντολών με συγκεκριμένη ευθύνη. Η function volume(radius, height) ορίζει τον τρόπο υπολογισμού. Η volume(2, 3) εκτελεί αυτόν τον τρόπο με συγκεκριμένες τιμές. Τα radius και height είναι παράμετροι του ορισμού· τα 2 και 3 είναι ορίσματα της κλήσης.

ΠΑΡΑΔΕΙΓΜΑ JS12, Ο ίδιος υπολογισμός με διαφορετικά δεδομένα

```

/** Δέχεται τιμή και ποσοστό. Επιστρέφει νέο ποσό χωρίς εμφάνιση.
 * Η αρχική τιμή δεν αλλάζει: ο καλών επιλέγει πού θα τη χρησιμοποιήσει.
 */
function discounted(price, percent) {
  return price * (1 - percent / 100);
}
const first = discounted(80, 25);
const second = discounted(120, 10);
console.log(first);
console.log(second);

```

Έξοδος της συγκεκριμένης δοκιμής

```

60
108

```

Το return παραδίδει το αποτέλεσμα στον κώδικα που έκανε την κλήση. Μόλις εκτελεστεί, η συγκεκριμένη κλήση της συνάρτησης ολοκληρώνεται.

Αν μια συνάρτηση κάνει μόνο console.log, δεν σημαίνει ότι επιστρέφει τον αριθμό που εμφάνισε. Χωρίς return η επιστροφή είναι undefined. Αυτό γίνεται πρόβλημα όταν θέλουμε να προσθέσουμε το αποτέλεσμα σε άλλο ποσό ή να το αποθηκεύσουμε στο ιστορικό. Κρατάμε συνήθως χωριστές συναρτήσεις εισόδου, υπολογισμού και αναφοράς.

ΠΑΡΑΔΕΙΓΜΑ JS13, Μια συνάρτηση καλεί μια άλλη

```

function baseArea(radius) {
  return Math.PI * radius ** 2;
}
function volume(radius, height) {
  // Επαναχρησιμοποιούμε την έννοια «εμβαδόν βάσης».
  return baseArea(radius) * height;
}
console.log(volume(1, 2).toFixed(3));

```

Έξοδος της συγκεκριμένης δοκιμής

```

6.283

```

Η volume δεν χρειάζεται να ξαναγράψει τον τύπο της βάσης. Η κλήση baseArea(1) επιστρέφει πρώτα π και μετά πολλαπλασιάζεται με 2.

Λειτουργία	Είσοδος	Επιστροφή	Εμφανίζει;
readNumber	Μήνυμα και όρια	Έναν αποδεκτό αριθμό	Πλαίσιο ερώτησης
volume	Ακτίνα και ύψος	Όγκο	Όχι
showReport	Αποτελέσματα	Δεν απαιτείται τιμή	Ναι, στην κονσόλα

Οι μεταβλητές που δημιουργούνται μέσα στη συνάρτηση είναι τοπικές στη συγκεκριμένη κλήση. Η χρήση παραμέτρων κάνει τις εξαρτήσεις φανερές: βλέπουμε τι χρειάζεται ο υπολογισμός. Μια συνάρτηση που διαβάζει κρυφά πολλές εξωτερικές μεταβλητές είναι δυσκολότερο να ελεγχθεί. Μην μεταφέρετε όμως κάθε απλή έκφραση σε ξεχωριστή συνάρτηση χωρίς λόγο· επιλέγετε ενότητες με σαφή έννοια.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γράψτε συνάρτηση για μετατροπή μέτρων σε εκατοστά. Πού πρέπει να βρίσκεται η console.log;

Απάντηση και αιτιολόγηση, function toCentimeters(meters) { return meters * 100; } Η εμφάνιση μπορεί να γίνει έξω: console.log(toCentimeters(1.5)); και δίνει 150. Έτσι η ίδια επιστροφή μπορεί να χρησιμοποιηθεί και σε άλλο τύπο.

Τεκμηρίωση κεφαλαίου: [10].

08 | Πίνακες: πολλές τιμές με σωστή σειρά

Στόχος, Να καταχωρίζετε, να διαβάζετε και να αλλάζετε στοιχεία χωρίς να χάνετε τη σχέση θέσης και δεδομένου.

Ένας πίνακας κρατά πολλές τιμές με σειρά. Ο πίνακας `grades = [6, 8, 10]` έχει μήκος 3, αλλά θέσεις 0, 1 και 2. Το `grades[1]` είναι ο δεύτερος βαθμός. Όταν ο χρήστης λέει «ο φοιτητής 1», συνήθως πρέπει να μετατρέψουμε τον ανθρώπινο αριθμό σε θέση 0, αφαιρώντας 1.

ΠΑΡΑΔΕΙΓΜΑ JS14, Διόρθωση δεν σημαίνει προσθήκη

```
const grades = [6, 8, 10];
const studentNumber = 2;
// Αλλάζουμε υπάρχουσα θέση, όχι το πλήθος των φοιτητών.
grades[studentNumber - 1] = 9;
grades.push(7);
console.log(grades.join(", "));
console.log("Πλήθος:", grades.length);
const last = grades.pop();
console.log("Αφαιρέθηκε:", last);
```

Έξοδος της συγκεκριμένης δοκιμής

6, 9, 10, 7
Πλήθος: 4
Αφαιρέθηκε: 7

Η ανάθεση στη θέση 1 διορθώνει το 8 σε 9. Η `push` προσθέτει τέταρτη εγγραφή. Η `pop` αφαιρεί και επιστρέφει την τελευταία. Η `const` επιτρέπει αυτές τις αλλαγές περιεχομένου.

Εργαλείο	Χρήση
<code>array.length</code>	Πόσα στοιχεία υπάρχουν.
<code>array.push(value)</code>	Προσθήκη στο τέλος.
<code>array.pop()</code>	Αφαίρεση και επιστροφή τελευταίου.
<code>array.includes(value)</code>	Έλεγχος αν υπάρχει συγκεκριμένη τιμή.
<code>array.slice()</code>	Νέος πίνακας με τα ίδια στοιχεία.
<code>array.join(", ")</code>	Ένα κείμενο που ενώνει τα στοιχεία.
<code>array.shift()</code>	Αφαίρεση πρώτου στοιχείου· χρήσιμη σε μικρά ιστορικά.

Ίδιος πίνακας ή αντίγραφο;

ΠΑΡΑΔΕΙΓΜΑ JS15, Δύο ονόματα μπορεί να δείχνουν το ίδιο αντικείμενο

```
const original = [4, 6];
const same = original;
const copy = original.slice();
same[0] = 9; // Αλλάζει ο πίνακας στον οποίο δείχνει και το original.
copy[1] = 8; // Αλλάζει ο νέος πίνακας, όχι το original.
console.log("Αρχικός:", original.join(", "));
console.log("Αντίγραφο:", copy.join(", "));
```

Έξοδος της συγκεκριμένης δοκιμής

Αρχικός: 9, 6
Αντίγραφο: 4, 8

Η ανάθεση `same = original` δεν αντιγράφει τα στοιχεία. Η `slice` δημιουργεί νέο εξωτερικό πίνακα. Αν τα στοιχεία είναι αντικείμενα, οι αναφορές σε αυτά εξακολουθούν να είναι κοινές: πρόκειται για ρηχό αντίγραφο.

Για μικρά ιστορικά μπορούμε να προσθέτουμε στο τέλος και, όταν το μήκος ξεπεράσει το όριο, να αφαιρούμε το πρώτο στοιχείο. Η επιλογή αυτή είναι κατανοητή και επαρκής για λίγες εγγραφές. Δεν χρειάζεται μια πιο δύσκολη δομή μόνο και μόνο για να φαίνεται ο κώδικας προχωρημένος.

Δοκιμάστε πριν διαβάσετε την απάντηση, Πώς κρατάμε τις τρεις τελευταίες τιμές ενός ιστορικού μετά από κάθε νέα προσθήκη;

**Απάντηση και αιτιολόγηση, `history.push(value)`; έπειτα `if (history.length > 3) history.shift()`;
Η διαγραφή γίνεται μόνο όταν ξεπεραστεί το όριο, όχι κάθε φορά.**

Τεκμηρίωση κεφαλαίου: [11], [2].

09 | Αλγόριθμοι πάνω σε πίνακες

Στόχος, Να υπολογίζετε σύνοψη, να επιλέγετε στοιχεία και να ταξινομείτε αντίγραφα με σωστό κριτήριο.

Πολλές εφαρμογές μοιράζονται τις ίδιες ιδέες: περνάμε από έναν πίνακα, μετράμε στοιχεία που πληρούν έναν κανόνα και κρατάμε κάποιο καλύτερο αποτέλεσμα. Πρώτα μαθαίνουμε τον ρητό βρόχο. Οι σύντομες μέθοδοι `map`, `filter` και `reduce` έχουν νόημα όταν μπορούμε ήδη να εξηγήσουμε τι θα έκανε ο αντίστοιχος βρόχος.

ΠΑΡΑΔΕΙΓΜΑ JS16, Μέσος όρος και στοιχεία αυστηρά πάνω από αυτόν

```
const values = [9, 9, 6, 4];
let sum = 0;
for (const value of values) sum += value;
const average = sum / values.length;
const above = [];
// Το δεύτερο πέρασμα γίνεται αφού έχει βρεθεί ο τελικός μέσος.
for (const value of values) {
  if (value > average) above.push(value);
}
console.log("Μέσος:", average);
console.log("Πάνω από τον μέσο:", above.join(", "));
```

Έξοδος της συγκεκριμένης δοκιμής

Μέσος: 7
Πάνω από τον μέσο: 9, 9

Δεν μπορούμε να επιλέξουμε σωστά τα στοιχεία χρησιμοποιώντας έναν μέσο όρο που αλλάζει ακόμη. Επίσης διατηρούμε και τα δύο 9: δεν υπάρχει λόγος να αφαιρεθεί η ισοβαθμία.

Ταξινόμηση αριθμών και διάμεσος

Η `sort` χωρίς συνάρτηση σύγκρισης ταξινομεί με βάση τη γραφή των τιμών, όχι με το αριθμητικό τους μέγεθος. Για αριθμητική αύξουσα σειρά δίνουμε σύγκριση `a - b`. Η `sort` αλλάζει τον πίνακα στον οποίο καλείται. Όταν χρειαζόμαστε και τη σειρά εισαγωγής, ταξινομούμε ένα αντίγραφο.

ΠΑΡΑΔΕΙΓΜΑ JS17, Διάμεσος για περιττό και άρτιο πλήθος

```
/** Επιστρέφει τη διάμεσο χωρίς να αλλάζει την αρχική σειρά. */
function median(values) {
  if (values.length === 0) throw new Error("Δεν υπάρχουν τιμές.");
  const ordered = values.slice();
  ordered.sort(function (a, b) { return a - b; });
  const middle = Math.floor(ordered.length / 2);
  if (ordered.length % 2 === 1) return ordered[middle];
  return (ordered[middle - 1] + ordered[middle]) / 2;
}
console.log(median([10, 2, 6]));
console.log(median([10, 2, 6, 8]));
```

Έξοδος της συγκεκριμένης δοκιμής6
7

Για [2, 6, 8, 10] δεν υπάρχει ένα μοναδικό μεσαίο στοιχείο. Παίρνουμε τον μέσο των 6 και 8. Δεν ταυτίζουμε τη διάμεσο με τον μέσο όλων των τιμών.

ΠΑΡΑΔΕΙΓΜΑ JS18, Μια δεύτερη γραφή με callbacks

```
const grades = [4, 7, 9];
// Η συνάρτηση μέσα στη filter καλείται για κάθε στοιχείο.
const passed = grades.filter(function (grade) { return grade >= 5; });
const total = grades.reduce(function (sum, grade) {
  return sum + grade;
}, 0);
console.log(passed.join(", "));
console.log(total);
```

Έξοδος της συγκεκριμένης δοκιμής7, 9
20

Callback είναι συνάρτηση που δίνουμε σε μια άλλη για να την καλέσει όταν χρειάζεται. Το 0 είναι η αρχική τιμή του αθροιστή. Η γραφή `grade => grade >= 5` είναι συντομότερη μορφή για αυτή την απλή συνάρτηση.

Προϋποθέσεις, Στο παράδειγμα υποθέτουμε ήδη ελεγμένους αριθμούς. Σε πραγματικό πρόγραμμα ο έλεγχος γίνεται πριν από την καταχώριση. Ένας άδειος πίνακας δεν έχει μέσο όρο ή διάμεσο που να ορίζεται αυτόματα ως 0.

Δοκιμάστε πριν διαβάσετε την απάντηση, Για [5,5,5], πόσα στοιχεία είναι πάνω από τον μέσο και πόσα μοιράζονται το μέγιστο;

Απάντηση και αιτιολόγηση, Κανένα δεν είναι αυστηρά πάνω από τον μέσο 5. Και τα τρία μοιράζονται το μέγιστο 5. Είναι διαφορετικά ερωτήματα και χρειάζονται διαφορετικοί έλεγχοι.

Τεκμηρίωση κεφαλαίου: [11].

10 | Αντικείμενα, ιδιότητες και μέθοδοι

Στόχος, Να κρατάτε μαζί τα στοιχεία μιας οντότητας και να συνδέετε δεδομένα με λειτουργίες.

Όταν κρατάμε το όνομα σε έναν πίνακα και τον βαθμό σε άλλον, πρέπει συνεχώς να θυμόμαστε ότι οι ίδιες θέσεις αντιστοιχούν στο ίδιο άτομο. Ένα αντικείμενο ενώνει αυτές τις σχετικές τιμές: `{name: "Άννα", grade: 8}`. Οι `name` και `grade` είναι ιδιότητες. Μια συνάρτηση αποθηκευμένη ως ιδιότητα λέγεται μέθοδος.

ΠΑΡΑΔΕΙΓΜΑ JS19, Πίνακας οργανωμένων εγγραφών

```
// Κάθε εγγραφή κρατά μαζί όνομα και βαθμό. Έτσι ο βρόχος δεν χρειάζεται
// να συντονίζει δύο διαφορετικούς πίνακες.
const students = [
  {name: "Άννα", grade: 8},
  {name: "Πέτρος", grade: 6}
];
for (const student of students) {
  console.log(student.name + ": " + student.grade);
}
```

Έξοδος της συγκεκριμένης δοκιμής

Άννα: 8

Πέτρος: 6

Ο βρόχος παίρνει μία ολόκληρη εγγραφή κάθε φορά. Η `students[0].grade` διαβάζει τον βαθμό της πρώτης εγγραφής. Δεν δημιουργούνται αυτόματα μεταβλητές με τα ονόματα των ιδιοτήτων.

ΠΑΡΑΔΕΙΓΜΑ JS20, Ένα αντικείμενο που περιγράφει κυλινδρική δεξαμενή

```
function createTank(radius, height) {
  return {
    radius, height,
    volume() { return Math.PI * this.radius ** 2 * this.height; },
    capacityLiters() { return this.volume() * 1000; }
  };
}
const tank = createTank(1, 2);
console.log(tank.volume().toFixed(3));
console.log(tank.capacityLiters().toFixed(1));
```

Έξοδος της συγκεκριμένης δοκιμής

```
6.283
6283.2
```

Το `this` αναφέρεται στο αντικείμενο `tank` από το οποίο γίνεται η κλήση. Η μέθοδος `volume()` υπολογίζει τον όγκο χωρίς να αλλάζει την ακτίνα ή το ύψος, ενώ η `capacityLiters()` επαναχρησιμοποιεί το αποτέλεσμα και αλλάζει μόνο τη μονάδα μέτρησης.

Η οργάνωση σε αντικείμενο είναι προαιρετική για το Θέμα Δ. Χρησιμοποιείται εδώ μόνο για να φανεί πώς σχετικές τιμές και λειτουργίες μπορούν να παραμένουν συγκεντρωμένες.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί είναι επικίνδυνο ένα δημόσιο `circle.radius = -4`;

Απάντηση και αιτιολόγηση, Η απευθείας ανάθεση μπορεί να παρακάμψει τον αρχικό έλεγχο. Μια μέθοδος `area` θα χρησιμοποιήσει πλέον μη έγκυρη ακτίνα. Η ελεγχόμενη αλλαγή με `setter` αντιμετωπίζει αυτό το πρόβλημα στα επόμενα κεφάλαια.

Τεκμηρίωση κεφαλαίου: [12].

11 | Έλεγχος εισόδου και διαχείριση σφαλμάτων

Στόχος, Να κατασκευάζετε αναγνώστη αριθμών που επαναλαμβάνει τα λάθη αλλά σέβεται την Ακύρωση.

Ο έλεγχος εισόδου έχει δύο μέρη. Πρώτα εξετάζουμε αν η γραφή είναι αποδεκτή. Έπειτα εξετάζουμε αν ο αριθμός βρίσκεται μέσα στα όρια. Η τιμή 11 είναι σωστά γραμμένος αριθμός, αλλά όχι αποδεκτός βαθμός όταν το όριο είναι 10. Η τιμή "5abc" αποτυγχάνει ήδη στο πρώτο μέρος.

Η `throw` δηλώνει ότι μια λειτουργία δεν μπορεί να ολοκληρωθεί κανονικά. Η `try` εκτελεί ένα τμήμα που ενδέχεται να αποτύχει. Η `catch` αποφασίζει τι θα κάνουμε με το σφάλμα. Στον αναγνώστη, λάθος αριθμός σημαίνει «ξαναρώτησε». Η Ακύρωση σημαίνει «σταμάτησε τη συνεδρία». Δεν πρέπει να βάλουμε και τα δύο στο ίδιο `catch` που επαναλαμβάνει χωρίς τέλος.

ΠΑΡΑΔΕΙΓΜΑ JS21, Αυστηρή δεκαδική γραφή και επανάληψη εισόδου

```
/** Μετατρέπει απλή δεκαδική γραφή. Δεν εμφανίζει τίποτε.
 * Επιτρέπει αρχικό πρόσημο και μία τελεία ή ένα κόμμα.
 */
function parseDecimal(raw, minimum, maximum) {
  const text = raw.trim().replace(",", ".");
  let digits = 0;
  let dots = 0;
  for (let i = 0; i < text.length; i++) {
    const character = text[i];
    if (character >= "0" && character <= "9") {
```

```

    digits++;
  } else if (character === ".") {
    dots++;
  } else if (i === 0 && (character === "+" || character === "-")) {
    // Το πρόσημο επιτρέπεται μόνο στην πρώτη θέση.
  } else {
    throw new Error("Χρησιμοποιήστε απλή δεκαδική γραφή.");
  }
}
const value = Number(text);
if (digits === 0 || dots > 1 || !Number.isFinite(value)) {
  throw new Error("Δεν δόθηκε έγκυρος αριθμός.");
}
if (value < minimum || value > maximum) {
  throw new Error(`Επιτρεπτά όρια: ${minimum} έως ${maximum}.`);
}
return value;
}
// Δέχεται μήνυμα και όρια. Επιστρέφει έγκυρο αριθμό· η Ακύρωση αφήνεται
// να φτάσει στον εξωτερικό χειρισμό.
function readNumber(message, minimum, maximum) {
  while (true) {
    const raw = prompt(message);
    if (raw === null) throw new Error("Η εκτέλεση ακυρώθηκε.");
    try {
      return parseDecimal(raw, minimum, maximum);
    } catch (error) {
      console.log(error.message);
      alert(error.message);
    }
  }
}
try {
  const grade = readNumber("Βαθμός 0-10:", 0, 10);
  console.log("Αποδεκτός βαθμός:", grade);
} catch (error) {
  console.log(error.message);
}

```

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

Έξοδος της συγκεκριμένης δοκιμής

Επιτρεπτά όρια: 0 έως 10.
Αποδεκτός βαθμός: 8.5

Τα δεδομένα δοκιμής είναι 11 και έπειτα 8,5. Το πρώτο catch αφορά μόνο τη μετατροπή. Το null δηλώνεται έξω από αυτό και φτάνει στο εξωτερικό catch, όπου ολοκληρώνεται η εκτέλεση.

Είσοδος	Απόφαση	Αιτία
" " ή "+"	Απόρριψη	Δεν υπάρχει ψηφίο.
"2.3.4"	Απόρριψη	Περισσότερες από μία τελείες.
"1e2" ή "0x10"	Απόρριψη	Δεν είναι απλή δεκαδική γραφή της άσκησης.
"8,5"	Αποδοχή	Το κόμμα μετατρέπεται σε τελεία.
null από prompt	Τερματισμός	Ρητή Ακύρωση, όχι λανθασμένος αριθμός.

Για ένα πεδίο που ζητά ακέραιο προσθέτουμε `Number.isInteger(value)`. Δεν χρησιμοποιούμε `parseInt` για να «διορθώσουμε» το 2.5 σε 2. Ο χρήστης πρέπει να γνωρίζει ότι η αρχική απάντηση απορρίφθηκε. Η συνάρτηση ελέγχου δεν αλλάζει την πρόθεση της εισόδου κρυφά.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί το `Number("")` χρειάζεται ξεχωριστή αντιμετώπιση;

Απάντηση και αιτιολόγηση, Επιστρέφει 0. Χωρίς έλεγχο της γραφής, ένα κενό πεδίο βαθμού μπορεί να καταχωριστεί ως πραγματικός μηδενικός βαθμός, ενώ ο χρήστης δεν έδωσε τίποτε.

Τεκμηρίωση κεφαλαίου: [J8], [J3], [J16].

12 | Μαθηματικά προβλήματα με σαφή βήματα

Στόχος, Να μεταφέρετε γεωμετρικούς τύπους και περιπτώσεις εξίσωσης σε μικρές ελέγξιμες συναρτήσεις.

Η σωστή λύση δεν αρχίζει από έναν μεγάλο τύπο. Αρχίζει από τις περιπτώσεις στις οποίες ο τύπος επιτρέπεται. Για την $ax^2+bx+c=0$ εξετάζουμε πρώτα το a . Αν $a=0$ δεν επιτρέπεται διαίρεση με $2a$. Αν $a \neq 0$, το πρόσημο της διακρίνουσας καθορίζει αν επιτρέπεται πραγματική τετραγωνική ρίζα.

Περίπτωση	Συμπέρασμα
$a=0, b \neq 0$	Γραμμική εξίσωση, $x=-c/b$.
$a=b=c=0$	Κάθε πραγματικός αριθμός είναι λύση.
$a=b=0, c \neq 0$	Καμία λύση.
$a \neq 0, \Delta=b^2-4ac > 0$	Δύο διαφορετικές πραγματικές ρίζες.
$a \neq 0, \Delta=0$	Μία διπλή πραγματική ρίζα.
$a \neq 0, \Delta < 0$	Καμία πραγματική ρίζα.

ΠΑΡΑΔΕΙΓΜΑ JS22, Έλεγχος γνωστών ριζών με αντικατάσταση

```
function valueAt(a, b, c, x) {
  // Ο ίδιος υπολογισμός χρησιμοποιείται για ρίζες και κορυφή.
  return a * x * x + b * x + c;
}
const a = 1, b = -5, c = 6;
const delta = b * b - 4 * a * c;
const root1 = (-b - Math.sqrt(delta)) / (2 * a);
const root2 = (-b + Math.sqrt(delta)) / (2 * a);
const vertexX = -b / (2 * a);
console.log(root1, valueAt(a, b, c, root1));
console.log(root2, valueAt(a, b, c, root2));
console.log(vertexX, valueAt(a, b, c, vertexX));
```

Έξοδος της συγκεκριμένης δοκιμής

```
2 0
3 0
2.5 -0.25
```

Το συγκεκριμένο παράδειγμα χρησιμοποιεί γνωστά έγκυρα δεδομένα με $a \neq 0$ και $\Delta > 0$. Δεν είναι γενικός επιλυτής. Ο πίνακας περιπτώσεων εξηγεί ποιοι έλεγχοι χρειάζονται πριν δεχθούμε αυθαίρετα δεδομένα.

Οι ρίζες είναι θέσεις όπου η $f(x)$ γίνεται 0. Η κορυφή είναι το κατώτερο ή ανώτερο σημείο της παραβολής. Δεν είναι η ίδια έννοια. Η $f(x)=x^2+1$ έχει κορυφή (0,1), αλλά δεν μηδενίζεται για πραγματικό x . Σε πρόγραμμα με δεκαδικούς συντελεστές η αντιμετώπιση μιας πολύ μικρής Δ πρέπει να ακολουθεί τον ρητό κανόνα ανοχής της εκφώνησης.

ΠΑΡΑΔΕΙΓΜΑ JS23, Μετατροπή μοιρών σε ακτίνια

```
// Η συνάρτηση δέχεται μοίρες, αλλά η Math.sin απαιτεί ακτίνια. Η
// μετατροπή παραμένει μέσα στη συνάρτηση.
function sineDegrees(degrees) {
  return Math.sin(degrees * Math.PI / 180);
}
console.log(sineDegrees(30).toFixed(6));
console.log(Math.hypot(3, 4));
```

Έξοδος της συγκεκριμένης δοκιμής

```
0.500000
5
```

Η Math.sin δέχεται ακτίνια. Η Math.hypot(3,4) δίνει το μήκος της υποτείνουσας, δηλαδή 5, και χρησιμοποιείται για την απόσταση ενός σημείου από ένα κέντρο.

Περιορισμοί πριν από Math, Στη ρίζα ελέγχουμε μη αρνητικό αριθμό. Στη διαίρεση ελέγχουμε μη μηδενικό παρονομαστή. Στη δύναμη ελέγχουμε τους κανόνες που ορίζει η εφαρμογή για μηδενική ή αρνητική βάση. Δεν εμφανίζουμε NaN ή Infinity σαν αποδεκτό τελικό αποτέλεσμα.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί μια δεξαμενή 50% γεμάτη δεν έχει τη μισή ακτίνα;

Απάντηση και αιτιολόγηση, Το ποσοστό εφαρμόζεται στον διαθέσιμο όγκο ή στα λίτρα. Η ακτίνα είναι κατασκευαστικό χαρακτηριστικό της δεξαμενής και δεν αλλάζει όταν αφαιρείται νερό.

Τεκμηρίωση κεφαλαίου: [J15], [J16].

13 | Τυχαιότητα και κατάσταση παιχνιδιού

Στόχος, Να παράγετε ακέραιους μέσα στα όρια και να ελέγχετε παιχνίδια με αναπαραγωγίμες δοκιμές.

Η Math.random() δίνει τιμή από 0 μέχρι μικρότερη του 1. Για ακέραιο από min έως max, μαζί με τα άκρα, χρησιμοποιούμε Math.floor(Math.random()*(max-min+1))+min. Η πρόσθεση του 1 στο πλάτος του διαστήματος είναι ουσιαστική: από 1 έως 6 υπάρχουν έξι διαφορετικές τιμές.

ΠΑΡΑΔΕΙΓΜΑ JS24, Οριακές δοκιμές του τύπου χωρίς να περιμένουμε την τύχη

```
function randomInteger(minimum, maximum, random = Math.random) {
  return Math.floor(random() * (maximum - minimum + 1)) + minimum;
}
// Στη δοκιμή δίνουμε δική μας συνάρτηση για το τυχαίο κλάσμα.
console.log(randomInteger(1, 100, function () { return 0; }));
console.log(randomInteger(1, 100, function () { return 0.999; }));
console.log(randomInteger(1, 100, function () { return 0.72; }));
```

Έξοδος της συγκεκριμένης δοκιμής

```
1
100
73
```

Η προαιρετική παράμετρος random επιτρέπει έλεγχο των άκρων. Χωρίς τρίτο όρισμα χρησιμοποιείται κανονικά η Math.random. Δεν αλλάζουμε την πραγματική συνάρτηση σε όλο το περιβάλλον.

Σε παιχνίδι μαντέματος, ο στόχος παράγεται μία φορά πριν από τον βρόχο. Οι έγκυρες διαφορετικές μαντεψιές αποθηκεύονται σε πίνακα. Μια επανάληψη της ίδιας τιμής δεν πρέπει να μειώνει τις προσπάθειες, αν αυτός είναι ο κανόνας. Η σειρά ενεργειών είναι: έλεγχος τιμής, έλεγχος διπλοτύπου, προσθήκη, έλεγχος επιτυχίας, υπόδειξη.

ΠΑΡΑΔΕΙΓΜΑ JS25, Ιστορικό έγκυρων μαντεψιών χωρίς διπλοεγγραφές

```
const guesses = [];
function addGuess(value) {
  if (guesses.includes(value)) return false;
  guesses.push(value);
  return true;
}
console.log(addGuess(50));
console.log(addGuess(75));
console.log(addGuess(50));
console.log(guesses.join(", "));
```

Έξοδος της συγκεκριμένης δοκιμής

```
true
true
false
50, 75
```

Η επαναλαμβανόμενη μαντεψιά δεν προστίθεται δεύτερη φορά. Η ίδια λογική μπορεί να χρησιμοποιηθεί όταν θέλουμε να ενημερώσουμε τον χρήστη ότι μια τιμή έχει ήδη δοκιμαστεί, χωρίς να αλλάξουμε τον μυστικό αριθμό.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί η αποκάλυψη της ακριβούς απόστασης από τον στόχο μπορεί να χαλάσει το παιχνίδι;

Απάντηση και αιτιολόγηση, Δίνει περισσότερη πληροφορία από μια γενική υπόδειξη. Με λίγες τέτοιες απαντήσεις ο στόχος μπορεί να προσδιοριστεί άμεσα. Η έξοδος πρέπει να ακολουθεί τους κανόνες του παιχνιδιού, όχι να εμφανίζει όλες τις βοηθητικές μεταβλητές.

Τεκμηρίωση κεφαλαίου: [18], [11].

14 | Ένα πλήρες μενού μόνο στην κονσόλα

Στόχος, Να συνδέετε είσοδο, επιλογές, αποθήκευση και επιβεβαίωση σε ένα πρόγραμμα που τερματίζει σωστά.

Το μενού δεν είναι γραφικό κουμπί. Είναι κείμενο που αντιστοιχίζει επιλογές σε εργασίες. Το παράδειγμα κρατά μικρές σημειώσεις στη μνήμη: προσθήκη, προβολή, διαγραφή και έξοδος. Η είσοδος μενού διαβάζεται ως κείμενο, επομένως δεν χρειάζεται να μετατραπεί σε αριθμό για να συγκριθεί με "1" ή "0".

ΠΑΡΑΔΕΙΓΜΑ JS26, Πρόχειρο σημειώσεων με έλεγχο διαγραφής

```
// Τα δεδομένα ζουν μέσα στη main. Το μενού επαναλαμβάνεται χωρίς να
// ξαναδημιουργείται ο πίνακας σε κάθε επιλογή.
```

```
function main() {
  const notes = [];
  while (true) {
    console.log("1 Προσθήκη | 2 Προβολή | 3 Διαγραφή | 0 Έξοδος");
    const raw = prompt("Επιλογή: 1, 2, 3 ή 0");
    if (raw === null || raw.trim() === "0") break;
    const choice = raw.trim();
    if (choice === "1") {
      const text = prompt("Κείμενο σημείωσης:");
      if (text === null) break;
      if (text.trim() === "") {
        console.log("Δεν καταχωρίστηκε κενή σημείωση.");
        continue;
      }
      notes.push(text.trim());
      console.log("Η σημείωση αποθηκεύτηκε στη μνήμη.");
    } else if (choice === "2") {
      if (notes.length === 0) console.log("Δεν υπάρχουν σημειώσεις.");
```

```

for (let i = 0; i < notes.length; i++) {
  console.log(` ${i + 1}. ${notes[i]} `);
}
} else if (choice === "3") {
  // Το false αρνείται τη διαγραφή, δεν κλείνει την εφαρμογή.
  if (confirm("Να διαγραφούν όλες οι σημειώσεις;")) {
    notes.length = 0;
    console.log("Οι σημειώσεις διαγράφηκαν.");
  } else {
    console.log("Οι σημειώσεις διατηρήθηκαν.");
  }
} else {
  console.log("Επιλέξτε μία από τις εμφανιζόμενες επιλογές.");
}
}
console.log("Τέλος συνεδρίας.");
}
main();

```

Δοκιμή εισόδου, με τη σειρά: 1 → Έλεγχος μονάδων → 2 → 3 → 0.

Απαντήσεις confirm: Ακύρωση / Όχι.

Έξοδος της συγκεκριμένης δοκιμής

```

1 Προσθήκη | 2 Προβολή | 3 Διαγραφή | 0 Έξοδος
Η σημείωση αποθηκεύτηκε στη μνήμη.
1 Προσθήκη | 2 Προβολή | 3 Διαγραφή | 0 Έξοδος
1. Έλεγχος μονάδων
1 Προσθήκη | 2 Προβολή | 3 Διαγραφή | 0 Έξοδος
Οι σημειώσεις διατηρήθηκαν.
1 Προσθήκη | 2 Προβολή | 3 Διαγραφή | 0 Έξοδος
Τέλος συνεδρίας.

```

Δοκιμή: προσθήκη «Έλεγχος μονάδων», προβολή, αίτημα διαγραφής που απορρίπτεται και έξοδος. Η λίστα υπάρχει μόνο για τη συγκεκριμένη εκτέλεση.

Γιατί η δομή είναι χρήσιμη

Ο βρόχος δεν αντιγράφεται μετά από κάθε επιλογή. Όταν ολοκληρωθεί μια εργασία, η ροή επιστρέφει στην αρχή του ίδιου while. Η continue χρησιμοποιείται όταν η συγκεκριμένη προσθήκη δεν μπορεί να ολοκληρωθεί. Το break χρησιμοποιείται όταν πρέπει να τελειώσει όλη η συνεδρία.

Το ίδιο οργανωτικό σχέδιο μπορεί να χρησιμοποιηθεί για επαναλαμβανόμενες δοκιμές μιας δεξαμενής: διαβάζουμε νέα ακτίνα και ύψος, καλούμε τις συναρτήσεις υπολογισμού και επιστρέφουμε στο ίδιο μενού μόνο αφού ολοκληρωθεί η αναφορά. Μενού, δεδομένα και υπολογισμός παραμένουν διαφορετικές ευθύνες.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ποια δοκιμή αποδεικνύει ότι η άρνηση διαγραφής λειτουργεί;

Απάντηση και αιτιολόγηση. Προσθέτουμε μια σημείωση, επιλέγουμε διαγραφή και απαντούμε Ακύρωση στην confirm. Ύστερα επιλέγουμε προβολή: η σημείωση πρέπει να εμφανίζεται κανονικά.

Τεκμηρίωση κεφαλαίου: [3], [5], [8], [11].

15 | Εμβέλεια, αναφορές και κλεισίματα

Στόχος, Να καταλαβαίνετε από πού διαβάζει μια συνάρτηση τις τιμές της και πότε μια αλλαγή επηρεάζει άλλο τμήμα.

Εμβέλεια είναι η περιοχή του κώδικα στην οποία μπορούμε να χρησιμοποιήσουμε ένα όνομα. Μια let μέσα σε μπλοκ { ... } δεν είναι διαθέσιμη έξω από αυτό. Μια τοπική μεταβλητή συνάρτησης

δημιουργείται για τη συγκεκριμένη κλήση. Μια καθολική μεταβλητή βρίσκεται έξω από τις συναρτήσεις και μπορεί να χρησιμοποιηθεί από πολλά τμήματα, άρα χρειάζεται προσοχή.

ΠΑΡΑΔΕΙΓΜΑ JS27, Παράμετρος και εξωτερική κατάσταση

```
// Η πρώτη συνάρτηση διαβάσει εξωτερική μεταβλητή. Η δεύτερη παίρνει τη
// συγκεκριμένη ακτίνα ως όρισμα.
let outsideRadius = 2;
function areaFromOutside() {
  return Math.PI * outsideRadius ** 2;
}
function areaFromParameter(radius) {
  return Math.PI * radius ** 2;
}
console.log(areaFromOutside().toFixed(2));
outsideRadius = 3;
console.log(areaFromOutside().toFixed(2));
console.log(areaFromParameter(2).toFixed(2));
```

Έξοδος της συγκεκριμένης δοκιμής

```
12.57
28.27
12.57
```

Η πρώτη συνάρτηση άλλαξε αποτέλεσμα χωρίς να αλλάξει η κλήση της. Η δεύτερη δείχνει στην κλήση ποια ακτίνα χρησιμοποιεί. Στο παράδειγμα η `outsideRadius` είναι εξωτερική ως προς τη συνάρτηση· αν βρίσκεται μέσα σε προστατευτικό μπλοκ δεν είναι καθολική ολόκληρου του περιβάλλοντος.

Τι σημαίνει «μεταβιβάζεται μια αναφορά»

Όταν δίνουμε έναν πίνακα σε συνάρτηση, περνάμε την τιμή της αναφοράς στο ίδιο αντικείμενο. Η συνάρτηση μπορεί να αλλάξει τα στοιχεία εκείνου του πίνακα. Αν όμως αναθέσει στην τοπική της παράμετρο έναν νέο πίνακα, δεν αλλάζει ποιον πίνακα δείχνει η μεταβλητή του καλούντος. Γι' αυτό τεκμηριώνουμε αν μια συνάρτηση τροποποιεί τα δεδομένα που δέχεται.

ΠΑΡΑΔΕΙΓΜΑ JS28, Μικρό κλείσιμο με ελεγχόμενη πρόσβαση

```
// Κάθε κλήση δημιουργεί ξεχωριστό count. Οι επιστρεφόμενες μέθοδοι
// διατηρούν πρόσβαση στο δικό τους count.
function createCounter() {
  let count = 0;
  return {
    increment() {
      count++;
      return count;
    },
    value() {
      return count;
    }
  };
}
const first = createCounter();
// Ο δεύτερος μετρητής ξεκινά ανεξάρτητα από τον πρώτο.
const second = createCounter();
console.log(first.increment());
console.log(first.increment());
console.log(second.value());
```

Έξοδος της συγκεκριμένης δοκιμής

```
1
2
0
```

Οι μέθοδοι συνεχίζουν να έχουν πρόσβαση στην count της κλήσης που τις δημιουργήσε. Αυτό λέγεται κλείσιμο (closure). Κάθε createCounter δημιουργεί δική του count, γι' αυτό ο δεύτερος μετρητής παραμένει 0.

Προαιρετική εμβάθυνση, Τα κλεισίματα είναι χρήσιμα για ελεγχόμενη πρόσβαση, αλλά δεν χρειάζονται σε κάθε αρχάριο πρόγραμμα. Ξεκινήστε με απλές παραμέτρους και αντικείμενα. Επιστρέψτε εδώ όταν χρειαστεί να εξηγήσετε τον προαιρετικό τρόπο κατασκευής κύκλου με τοπικά δεδομένα.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί μια συνάρτηση που υπολογίζει μέσο όρο δεν πρέπει να ταξινομεί κρυφά τον αρχικό πίνακα;

Απάντηση και αιτιολόγηση, Ο υπολογισμός του μέσου δεν απαιτεί αλλαγή σειράς. Μια κρυφή ταξινόμηση μπορεί να χαλάσει την αντιστοίχιση με τη σειρά καταχώρισης. Είτε δεν μεταβάλλουμε τον πίνακα είτε το δηλώνουμε ρητά ως μέρος της λειτουργίας.

Τεκμηρίωση κεφαλαίου: [J2], [J10], [J12].

16 | Κλάσεις και ελεγχόμενες ιδιότητες

ΠΡΟΑΙΡΕΤΙΚΗ ΕΜΒΑΘΥΝΣΗ

Στόχος, Να χρησιμοποιείτε ένα κοινό πρότυπο αντικειμένων και να προστατεύετε την εγκυρότητα των τιμών.

Μια κλάση περιγράφει ποια στοιχεία και ποιες λειτουργίες θα έχουν τα αντικείμενα που δημιουργούνται από αυτή. Η new δημιουργεί συγκεκριμένο αντικείμενο. Η constructor το αρχικοποιεί. Δεν είναι αναγκαίο κάθε απλό πρόγραμμα να χρησιμοποιεί κλάσεις: έχουν νόημα όταν υπάρχουν πολλές οντότητες με κοινή συμπεριφορά.

ΠΑΡΑΔΕΙΓΜΑ JS29, Απόθεμα που δεν γίνεται αρνητικό

```
class Supply {
  #quantity;
  constructor(name, quantity) {
    this.name = name;
    this.quantity = quantity; // Χρησιμοποιεί και τον αρχικό έλεγχο.
  }
  get quantity() {
    return this.#quantity;
  }
  // Πρώτα ελέγχουμε και μετά αποθηκεύουμε, ώστε λάθος ανάθεση να μη
  // χαλάσει την κατάσταση.
  set quantity(value) {
    if (!Number.isInteger(value) || value < 0) {
      throw new RangeError("Χρειάζεται μη αρνητικό ακέραιο απόθεμα.");
    }
    // Η αποθήκευση ακολουθεί τον έλεγχο, όχι το αντίστροφο.
    this.#quantity = value;
  }
  take(amount) {
    if (!Number.isInteger(amount) || amount <= 0) {
      throw new RangeError("Η ποσότητα πρέπει να είναι θετική.");
    }
    if (amount > this.#quantity) return false;
    this.#quantity -= amount;
    return true;
  }
}

const pens = new Supply("Στυλό", 3);
console.log(pens.take(2));
console.log(pens.take(2));
```

```
console.log("Υπόλοιπο:", pens.quantity);
try {
  pens.quantity = -5;
} catch (error) {
  console.log(error.message);
}
console.log("Μετά την άκυρη αλλαγή:", pens.quantity);
```

Έξοδος της συγκεκριμένης δοκιμής

```
true
false
Υπόλοιπο: 1
Χρειάζεται μη αρνητικό ακέραιο απόθεμα.
Μετά την άκυρη αλλαγή: 1
```

Ο setter χρησιμοποιείται σαν ανάθεση, όχι σαν κλήση συνάρτησης. Η άκυρη ανάθεση δεν αλλάζει το προηγούμενο απόθεμα. Η take επιστρέφει false όταν η επιθυμητή ποσότητα είναι θετική αλλά δεν επαρκεί το απόθεμα.

Το #quantity είναι ιδιωτικό στοιχείο της κλάσης. Δεν προσπελάζεται ως pens.#quantity από εξωτερικό κώδικα. Το _quantity, χωρίς #, θα ήταν απλώς σύμβαση ονόματος. Η ιδιωτικότητα βοηθά στην οργάνωση και στην αποφυγή μη επιτρεπτών αλλαγών· δεν κρυπτογραφεί δεδομένα και δεν αποτελεί από μόνη της μηχανισμό ασφάλειας συστήματος.

Έννοια	Πρακτική ανάγνωση
constructor	Τι πρέπει να γίνει όταν δημιουργείται το αντικείμενο.
this	Το συγκεκριμένο αντικείμενο της κλήσης.
get quantity	Τι δίνουμε όταν ζητηθεί ανάγνωση.
set quantity	Ποιοι έλεγχοι προηγούνται της αποθήκευσης.
Μέθοδος take	Μια ολοκληρωμένη ενέργεια με κανόνες.

Δοκιμάστε πριν διαβάσετε την απάντηση, Γιατί η constructor καλεί this.quantity = quantity αντί να γράψει κατευθείαν στο #quantity;

Απάντηση και αιτιολόγηση, Για να χρησιμοποιηθεί ο ίδιος έλεγχος κατά τη δημιουργία και σε μεταγενέστερες αλλαγές. Έτσι δεν υπάρχει δεύτερη διαδρομή αρχικοποίησης που παρακάμπτει τον κανόνα.

Τεκμηρίωση κεφαλαίου: [J13], [J14].

17 | Κείμενα, Set και οργανωμένες συλλογές

ΠΡΟΑΙΡΕΤΙΚΗ ΕΜΒΑΘΥΝΣΗ

Στόχος, Να χρησιμοποιείτε κείμενο και Set όταν χρειάζεται να θυμάστε μοναδικές επιλογές χωρίς να αλλοιώνετε τα αρχικά δεδομένα.

Στο παιχνίδι μαντεψιάς μια τιμή μπορεί να αποθηκεύεται ως αριθμός, ενώ τα μηνύματα προς τον χρήστη είναι κείμενα. Το Set είναι συλλογή μοναδικών τιμών και είναι χρήσιμο όταν θέλουμε να γνωρίζουμε γρήγορα αν μια μαντεψιά έχει ήδη εμφανιστεί.

ΠΑΡΑΔΕΙΓΜΑ JS30, Set με τις μαντεψιές του παίκτη

```
const used = new Set();
used.add(50);
used.add(75);
used.add(50);
console.log(used.size);
```

```
console.log(used.has(75));
console.log([...used].join(", "));
```

Έξοδος της συγκεκριμένης δοκιμής

```
2
true
50, 75
```

Η δεύτερη `add(50)` δεν αλλάζει το `Set`. Αν χρειάζεται να διατηρείται αυστηρά και η σειρά των προσπαθειών, ένας απλός πίνακας παραμένει πολύ κατανοητή λύση. Το `Set` παρουσιάζεται ως εναλλακτικό εργαλείο.

ΠΑΡΑΔΕΙΓΜΑ JS31, Ομαδοποίηση βαθμών χωρίς παράλληλους πίνακες

```
const students = [
  {name: "Άννα", grade: 8.5},
  {name: "Νίκος", grade: 6.5}
];
for (const student of students) {
  console.log(` ${student.name}: ${student.grade}`);
}
```

Έξοδος της συγκεκριμένης δοκιμής

```
Άννα: 8.5
Νίκος: 6.5
```

Κάθε αντικείμενο κρατά μαζί το όνομα και τον βαθμό. Έτσι η ταξινόμηση ή το φιλτράρισμα του Θέματος Α δεν χρειάζεται να συντονίζει πολλούς παράλληλους πίνακες.

Τεκμηρίωση κεφαλαίου: [17], [11].

18 | Αποσφαλμάτωση και δοκιμές που αποδεικνύουν κάτι

Στόχος, Να εντοπίζετε το είδος ενός λάθους και να σχεδιάζετε ελέγχους με γνωστή αναμενόμενη έξοδο.

Αποσφαλμάτωση είναι η συστηματική αναζήτηση της αιτίας ενός λάθους. Δεν σημαίνει ότι αλλάζουμε τυχαία εντολές μέχρι να εμφανιστεί ένας επιθυμητός αριθμός. Ξεκινάμε με την αναμενόμενη συμπεριφορά, κρατάμε ένα μικρό παράδειγμα που αποτυγχάνει και παρακολουθούμε τις ενδιάμεσες τιμές.

Είδος	Παράδειγμα	Πρώτη ενέργεια
Σύνταξη	Λείπει <code>}</code> ή κλείσιμο εισαγωγικών.	Διαβάστε θέση και είδος <code>SyntaxError</code> .
Εκτέλεση	Χρήση ονόματος που δεν έχει δηλωθεί.	Ελέγξτε ορθογραφία και εμβέλεια.
Λογική	Χρεώνεται δύο φορές μια ίδια μαντεψιά.	Παρακολουθήστε τη σειρά ελέγχων και αλλαγών.
Αριθμητική	Στρογγυλοποίηση πριν από σύγκριση.	Εκτυπώστε την αρχική αριθμητική τιμή.

ΠΑΡΑΔΕΙΓΜΑ JS32, Μικρές αυτοματοποιημένες προσδοκίες

```
// Οι έλεγχοι συγκρίνουν πραγματική και αναμενόμενη συμπεριφορά στα όρια
// των κατηγοριών. Έτσι εντοπίζονται λάθη στα σύμβολα σύγκρισης.
function category(grade) {
  if (grade >= 8.5) return "Άριστα";
  if (grade >= 6.5) return "Λίαν Καλώς";
  if (grade >= 5) return "Καλώς";
  return "Αποτυχία";
}
console.assert(category(8.5) === "Άριστα", "Όριο 8.5");
console.assert(category(5) === "Καλώς", "Όριο 5");
console.assert(category(4.99) === "Αποτυχία", "Κάτω από 5");
console.log("Οι τρεις έλεγχοι ολοκληρώθηκαν.");
```

Έξοδος της συγκεκριμένης δοκιμής

Οι τρεις έλεγχοι ολοκληρώθηκαν.

Η console.assert γράφει αποτυχία όταν η συνθήκη είναι false. Δεν είναι γενικός μηχανισμός εισόδου και δεν πρέπει να χρησιμοποιείται αντί για if/throw στις προδιαγραφές της εφαρμογής.

Πίνακας ελέγχων

Κατηγορία δοκιμής	Τι ελέγχουμε
Κανονική τιμή	Ένα χειροκίνητα υπολογισμένο αποτέλεσμα.
Κάτω και πάνω όριο	Ακριβώς το 0, το 10 ή το μέγιστο πλήθος.
Μη έγκυρη είσοδος	Κενό, κείμενο, εκτός ορίων, μη ακέραιο.
Ακύρωση	Τερματισμό χωρίς ατέρμονη επανάληψη.
Κατάσταση μετά από αποτυχία	Το απόθεμα ή το ιστορικό μένει αμετάβλητο.
Δεύτερη εκτέλεση	Δεν χρησιμοποιούνται παλιά δεδομένα από μνήμη.

Ένα breakpoint σταματά προσωρινά τον κώδικα πριν από μια εντολή, ώστε να εξετάσουμε τιμές. Ακόμη και χωρίς debugger μπορούμε να κάνουμε χειροκίνητη παρακολούθηση σε πίνακα. Οι προσωρινές εκτυπώσεις αφαιρούνται από την τελική λύση αν αποκαλύπτουν κρυφές τιμές παιχνιδιού ή μπερδεύουν την αναφορά.

Δοκιμάστε πριν διαβάσετε την απάντηση, Ένα πρόγραμμα δίνει σωστή απάντηση για 8/2. Αρκεί αυτή η δοκιμή;

Απάντηση και αιτιολόγηση, Όχι. Χρειάζονται τουλάχιστον παρονομαστής 0, αρνητικός αριθμός, δεκαδικό πηλίκο, λανθασμένη είσοδος και Ακύρωση. Η μία επιτυχής περίπτωση δεν ελέγχει τους κλάδους αποτυχίας.

Τεκμηρίωση κεφαλαίου: [J8], [J19], [J1].

19 | Ολοκληρωμένο παράδειγμα: μικρή αναφορά βαθμολογίας

Στόχος, Να συνδέσετε έλεγχο εισόδου, πίνακα αντικειμένων, συναρτήσεις και καθαρή αναφορά χωρίς να μπερδεύετε την είσοδο με τους υπολογισμούς.

Το παράδειγμα χρησιμοποιεί τρεις φοιτητές για να παραμείνει μικρό. Η ίδια δομή επεκτείνεται σε N φοιτητές: η είσοδος κατασκευάζει τις εγγραφές, η analyze() υπολογίζει τα στατιστικά και η printReport() παρουσιάζει τα αποτελέσματα.

ΠΑΡΑΔΕΙΓΜΑ JS33, Πλήρης μικρή αναφορά βαθμολογίας

```

function category(grade) {
  if (grade >= 8.5) return "Άριστα";
  if (grade >= 6.5) return "Λίαν Καλώς";
  if (grade >= 5) return "Καλώς";
  return "Αποτυχία";
}
function analyze(students) {
  let sum = 0;
  for (const student of students) sum += student.grade;
  const average = sum / students.length;
  const above = students.filter(s => s.grade > average);
  return {average, above};
}
function printReport(students, result) {
  for (const s of students) console.log(` ${s.name}: ${s.grade} - ${category(s.grade)} `);
  console.log(` Μέσος όρος: ${result.average.toFixed(2)} `);
}

```

```

console.log("Πάνω από τον μέσο όρο:", result.above.map(s => s.name).join(", "));
}
const students = [
  {name: "Άννα", grade: 9},
  {name: "Νίκος", grade: 7},
  {name: "Μαρία", grade: 4}
];
printReport(students, analyze(students));

```

Έξοδος της συγκεκριμένης δοκιμής

Άννα: 9 - Άριστα
 Νίκος: 7 - Λίαν Καλώς
 Μαρία: 4 - Αποτυχία
 Μέσος όρος: 6.67
 Πάνω από τον μέσο όρο: Άννα, Νίκος

Παρακολούθηση κατάστασης

Η `analyze()` δεν αλλάζει τον πίνακα. Παράγει νέο αποτέλεσμα από τα δεδομένα που της δόθηκαν. Η `printReport()` δεν ξαναυπολογίζει τον μέσο όρο, αλλά χρησιμοποιεί την οργανωμένη επιστροφή της `analyze()`.

Δοκιμάστε πριν διαβάσετε την απάντηση, Τι πρέπει να συμβεί αν όλοι οι βαθμοί είναι ίσοι;

Απάντηση και αιτιολόγηση, Ο μέσος όρος είναι ίσος με κάθε βαθμό και η λίστα `above` είναι κενή, επειδή ζητάμε αυστηρά μεγαλύτερο βαθμό από τον μέσο όρο.

Τεκμηρίωση κεφαλαίου: [10], [12], [8].

20 | Από τη μελέτη στην υλοποίηση των θεμάτων

Στόχος, Να οργανώνετε τη μελέτη και να ελέγχετε μόνοι σας αν είστε έτοιμοι να κατασκευάσετε ολόκληρο πρόγραμμα.

Η υλοποίηση μιας άσκησης αρχίζει με μια μικρή περιγραφή στο χαρτί. Γράψτε είσοδο, όρια, πράξεις, ειδικές περιπτώσεις και έξοδο. Στη συνέχεια δοκιμάστε τις συναρτήσεις υπολογισμού με σταθερά δεδομένα. Συνδέστε τα πλαίσια εισόδου τελευταία, ώστε ένα πρόβλημα ανάγνωσης να μη συγχέεται με λανθασμένο μαθηματικό τύπο.

Περιγραφή του Προβλ.	Κατηγορία προβλημάτων	Τι πρέπει να μπορεί να ελέγξει
A · Εικόνα τμήματος	2-9, 11, 18	Δύο περάσματα, κατηγορίες, ισοβαθμίες και ποσοστό.
B · Εξίσωση και επαλήθευση	3, 5, 7, 11-12	Γιατί προηγείται το $a=0$ και τι ελέγχει η αντικατάσταση.
Γ · Μάντεψε με στρατηγική	4-8, 11, 13, 17-18	Σταθερός στόχος, άκυρες είσοδοι και έλεγχος επαναλαμβανόμενων μαντεψιών.
Δ · Δεξαμενή και επάρκεια νερού	3, 7, 10, 12, 15-16, 18	Μονάδες, όγκος, λίτρα, πλήρωση και έλεγχος παραμέτρων.

Πέντε έλεγχοι ετοιμότητας

1. Μπορώ να υπολογίσω ένα μικρό παράδειγμα με το χέρι και να γράψω την αναμενόμενη έξοδο.

- Μπορώ να εξηγήσω κάθε συνάρτηση με μία πρόταση: τι δέχεται, τι δίνει και αν αλλάζει δεδομένα.
- Ξέρω τι κάνει το πρόγραμμα σε λάθος είσοδο, σε Ακύρωση και σε μαθηματική αποτυχία.
- Οι εκτυπώσεις έχουν περιγραφές και μονάδες και δεν αποκαλύπτουν κρυφές πληροφορίες.
- Έχω δοκιμάσει όρια, κενές συλλογές όπου επιτρέπονται, ισοβαθμίες και δεύτερη εκτέλεση.

Πώς γράφω ουσιαστικό ελληνικό σχόλιο

Ένα σχόλιο «αυξάνει το i» πάνω από `i++` επαναλαμβάνει αυτό που ήδη βλέπουμε. Πιο χρήσιμο είναι: «Αυξάνουμε τις προσπάθειες μόνο μετά από έγκυρη μαντεψιά, ώστε οι άκυρες είσοδοι να μην επηρεάζουν το σκορ». Το δεύτερο εξηγεί τον λόγο. Στην αρχή συνάρτησης περιγράψτε τις παραμέτρους, την επιστροφή και τους κανόνες που προστατεύει. Αν αλλάξει ο κώδικας, ενημερώστε και το σχόλιο.

Σύντομη αναφορά

Ανάγκη	Εργαλείο
Αριθμός με δεκαδικά	Number και έλεγχος <code>Number.isFinite</code>
Ακέραιος	<code>Number.isInteger</code> μετά τη μετατροπή
Απόλυτη διαφορά	<code>Math.abs(a - b)</code>
Αντίγραφο πίνακα	<code>array.slice()</code>
Εύρεση τιμής	<code>array.includes(value)</code>
Έλεγχος ιδιότητας αντικειμένου	Σαφής έλεγχος κωδικών ή <code>Object.hasOwn</code>
Πρόωρο τέλος συνάρτησης	<code>return</code>
Προγραμματισμένη αποτυχία	<code>throw new Error(...)</code>
Χειρισμός αποτυχίας	<code>try / catch</code>
Στοιχεία μνήμης ανά οντότητα	Αντικείμενο ή κλάση
Μήνυμα που παραμένει	<code>console.log</code>

Δοκιμάστε πριν διαβάσετε την απάντηση, Πότε θεωρείται ολοκληρωμένη μια λύση;

Απάντηση και αιτιολόγηση, Όταν ικανοποιεί όλα τα ζητούμενα και τους κανόνες της εκφώνησης, όχι μόνο το εύκολο παράδειγμα. Πρέπει επίσης να εκτελείται αυτοτελώς, να εξηγείται και να μην αλλάζει κρυφά δεδομένα μετά από αποτυχία.

Πηγές και συμπληρωματική μελέτη

Οι πηγές παρακάτω τεκμηριώνουν τη συμπεριφορά της γλώσσας και των εργαλείων της. Τα εκπαιδευτικά σενάρια, οι τιμές παραδειγμάτων και οι κανόνες των εφαρμογών ορίζονται εδώ. Για λεπτομέρειες μιας συνάρτησης συμβουλευθείτε την αντίστοιχη επίσημη σελίδα, όχι ένα αποτέλεσμα αναζήτησης χωρίς συμφραζόμενα.

[1] [Chrome DevTools: εκτέλεση και αποθήκευση Snippets](#)

[3] [MDN: prompt και διάκριση κενού από Ακύρωση](#)

[5] [MDN: confirm](#)

[7] [MDN: εκφράσεις και τελεστές](#)

[2] [MDN: δηλώσεις, εμβέλεια, τιμές και τύποι](#)

[4] [MDN: alert](#)

[6] [MDN: console.log](#)

[8] [MDN: συνθήκες και διαχείριση σφαλμάτων](#)

[J9] MDN: επαναλήψεις

[J11] MDN: πίνακες

[J13] MDN: κλάσεις

[J15] MDN: μαθηματικές συναρτήσεις

[J17] MDN: Set και Map

[J19] MDN: console.assert

[J10] MDN: συναρτήσεις και κλεισίματα

[J12] MDN: αντικείμενα, ιδιότητες και μέθοδοι

[J14] MDN: ιδιωτικά στοιχεία κλάσεων

[J16] MDN: αριθμητικός τύπος Number

[J18] MDN: Math.random

[J20] MDN: String και χειρισμός κειμένου