

# Ghid de abordare:

## TEMA 1, strategii și reamintiri (JS, ES6+, paradigme)

Acest ghid oferă studenților direcții de lucru fără a furniza soluția efectivă. Structura urmează taxonomia Bloom și include strategii de testare, structură logică recomandată, capcane frecvente, o listă de verificare înainte de predare, precum și reamintiri ale unor detalii tehnice din JavaScript (funcții, obiecte, array-uri, variabile/constante) și discuția paradigmelor imperativ vs. funcțional.

### 1) ABORDARE TEMEI mapata pe nivelurile taxonomiei Bloom

#### 1.1. REMEMBER

##### Obiective:

- Rețineți definiția și scopul celor doi algoritmi: **RLE** (Run-Length Encoding) pentru compresie, **Cifrul Caesar** pentru criptare cu deplasare fixă.
- Memorați cerințele de validare: tipuri (``string`/`String``, ``boolean``), restricțiile de conținut (RLE: fără cifre la compresie; Caesar: doar litere și spații) și necesitatea ``options.shift`` întreg pentru Caesar.
- Înțelegeți semantica parametrului ``operation``: ``true`` ⇒ compresie/criptare; ``false`` ⇒ decompresie/decriptare.

##### Activități sugerate (fără cod):

- Scrieți 2–3 exemple scurte de intrări/ieșiri (șir gol, un singur caracter, text cu spații).
- Listați explicit erorile așteptate: ce nume de excepție și de ce este ridicată.

#### 1.2. UNDERSTAND

##### Obiective:

- Explicați de ce RLE funcționează (exploatează redundanța) și când nu aduce beneficii (date fără repetiții).
- Explicați **invarianta** cifrului Caesar: deplasări mod 26, păstrarea spațiilor, sensibilitatea la registru; decriptarea este deplasarea inversă.
- Distingeți între compresie (RLE) și criptare (Caesar): obiective, criterii de succes, metrice (raport de compresie versus confidențialitate).

##### Activități sugerate:

- Pentru 3 șiruri date, determinați mental efectul compresiei/decompresiei și al criptării/decriptării.
- Schițați fluxul de date: ``textProcessor`` → validări → funcții pentru algoritmi → rezultat.

#### 1.3. APPLY

##### Obiective:

- Aplicați validările ca precondiții înainte de logica algoritmilor.
- Aplicați o mini-disciplină TDD: formulați așteptări, rulați testele oficiale, interpretați mesajele.

##### Activități sugerate:

- Formulați cazuri de test sub formă de propoziții verificabile: „Dacă intrarea conține cifre la compresie RLE, se aruncă ``InvalidInput``” etc.
- Abordare incrementală: RLE-compresie → RLE-decompresie → Caesar-criptare → Caesar-decriptare.

## 1.4. ANALYSE

### Obiective:

- Descompuneți în module pure: „dispecer” (selectează algoritmul), validări comune, implementări izolate pentru RLE și Caesar.
- Analizați complexitatea: toate transformările să fie  $O(n)$  în lungimea șirului; notați și spațiul auxiliar.
- Comparați imperative vs. funcțional (avantaje ale compoziției și imutabilității versus lizibilitate în expresii dense).

### Activități sugerate:

- Faceți o diagramă a dependențelor dintre funcții.
- Identificați puncte de defectare: șir gol; numărări pe mai multe cifre la RLE; deplasări negative sau  $>26$  la Caesar; caractere nepermise la Caesar.

## 1.5. EVALUATE

### Obiective:

- Evaluați corectitudinea cu proprietăți:  $\text{decompresie}(\text{compresie}(x)) = x$ ;  $\text{decriptare}(\text{criptare}(x,s),s) = x$ .
- Evaluați calitatea designului: coeziune, separarea responsabilităților, consistența erorilor (``InvalidType`/`InvalidInput``).
- Argumentați alegerea stilului (imperativ vs. funcțional) în termeni de mentenabilitate și simplitate cognitivă.

### Activități sugerate:

- Revizuire în perechi pe baza unei grile scurte: corectitudine, lizibilitate, erori, simplitate algoritmică.

## 1.6. CREATE

- Integrați toate piesele într-o implementare coerentă cu interfața exactă ``textProcessor(algo, operation, input, options)``.
- Scrieți un README scurt: cum se rulează testele, constrângeri de intrare, erori ridicate și de ce.
- Dacă timpul permite: separați validările într-un modul reutilizabil și documentați-l prin exemple.

## 2) CE NI S-A PREDAT ... pana acum (JavaScript, ES6+, paradigme)

### 2.1. FUNCȚII

- Preferință pentru funcții pure, fără efecte secundare; semnături clare; fie returnează rezultat, fie aruncă eroare documentată.
- Funcții de utilitate mici și compozabile: p. ex. validări, normalizări, transformări de text.

### 2.2. OBIECTE & ARRAY-URI

- Tratați obiectul ``options`` defensiv: verificați existența și tipurile proprietăților necesare (``options.shift`` întreg la Caesar).
- Evitați mutațiile necontrolate; lucrați imutabil (copii/rezultate noi).

### 2.3. VARIABLE ȘI CONSTANTE

- ``const`` pentru referințe neschimbate; ``let`` doar când e strict necesar.
- Denumiri semantice, clare, pentru ușurința revizuirii.

### 2.4. IMPERATIV VS. FUNCȚIONAL

- Imperativ: clar atunci când aveți stări tranzitorii explicite.
- Funcțional: avantaje la prelucrări de string-uri (mapare, compoziție, expresii regulate), cu atenție la lizibilitate.

### 3) Strategie de TESTARE (fără a modifica testele!)

- Începeți cu validările: aduc feedback util și oferă rapid claritate.
- Ordine logică: RLE-compresie → RLE-decompresie → Caesar-criptare → Caesar-decriptare.
- Cazuri de margine: șir gol; un caracter; serii lungi; numărări cu mai multe cifre la RLE; spații multiple; deplasări negative/mari; caractere nepermise la Caesar.
- Proprietăți utile (conceptual):  $\text{rleDecompress}(\text{rleCompress}(x)) = x$  (pentru  $x$  fără cifre);  $\text{caesarDecrypt}(\text{caesarEncrypt}(x, s), s) = x$ .
- Design by Contract: verificați aruncarea ``InvalidType`` vs. ``InvalidInput`` la încălcări ale condițiilor.
- Ciclu scurt de lucru: red → green → refactor pe fiecare cerință.

### 4) STRUCTURĂ logică RECOMANDATĂ (fără cod)

- **Fațadă:** ``textProcessor(algo, operation, input, options)`` — normalizează ``algo``, verifică ``operation``, rutează către algoritm.
- **Validări comune:** tipurile corecte pentru ``algo``, ``operation``, ``input``.
- **RLE:** două funcții pure separate pentru compresie/decompresie, cu validările aferente.
- **Caesar:** două funcții pure; normalizare deplasare mod 26; păstrarea spațiilor și a registrului.
- **Tratarea erorilor:** o funcție simplă de creare a erorilor care setează ``name`` la ``InvalidType``/``InvalidInput``.

### 5) CAPCANE frecvente

- Confuzia între ``InvalidType`` și ``InvalidInput``: declarați clar când se folosește fiecare.
- RLE-decompresie: nu presupuneți numărări pe o singură cifră; acceptați și valori cu mai multe cifre.
- Caesar: nu acceptați caractere în afara [A–Z a–z] și spații; normalizați deplasarea (``s mod 26``), inclusiv pentru ``s`` negativ.
- Interfața de export: păstrați CommonJS (``module.exports``) dacă testele o cer.
- Nu modificați testele; citiți mesajele de eroare — ele indică condiția încălcată.

### 6) LISTĂ DE VERIFICARE înainte de predare

- Validări de tip și conținut implementate; erorile au ``name`` corect.
- RLE: compresie/decompresie corecte (inclusiv numărări cu mai multe cifre).
- Caesar: criptare/decriptare corecte; normalizare ``shift``; păstrarea spațiilor și a registrului.
- Complexitate liniară; fără costuri inutile.
- Testele oficiale: verzi; cazuri de margine acoperite conceptual.
- Cod curat: funcții mici, nume clare, fără duplicare.

### 7) OBSERVAȚII METODOLOGICE

- Stilul funcțional aduce compoziție și lipsa efectelor secundare; nu-l forțați dacă devine opac pentru echipă.
- TDD oferă „plasa de siguranță” și documentează specificația prin exemple.
- Testarea bazată pe proprietăți validează invariante generale, nu doar cazuri punctuale.

## REFERINȚE (doar pentru voluntari)

- Salomon, D. (2007). Data Compression: The Complete Reference (4th ed.). Springer. <https://doi.org/10.1007/978-1-84628-603-2>
- Sayood, K. (2017). Introduction to Data Compression (5th ed.). Morgan Kaufmann. <https://doi.org/10.1016/C2015-0-06248-7>
- Bauer, F. L. (2011). Cæsar Cipher. In H. C. A. van Tilborg & S. Jajodia (Eds.), Encyclopedia of Cryptography and Security. Springer. [https://doi.org/10.1007/978-1-4419-5906-5\\_162](https://doi.org/10.1007/978-1-4419-5906-5_162)