**1️⃣ Find products above average price**

**Briefing**: Given an array of product objects with name and price, calculate the average price and return products whose price is strictly greater than the average.

**Test Cases**:

* **Input**: [{name: "Laptop", price: 1000}, {name: "Phone", price: 500}, {name: "Tablet", price: 300}]
  + **Output**: [{name: "Laptop", price: 1000}] (Average=600)
* **Input**: [{name: "Book", price: 20}, {name: "Pen", price: 20}]
  + **Output**: []
* **Input**: []
  + **Output**: []

**2️⃣ Group users by subscription**

**Briefing**: Given users with name and type ("Pro"/"Free"), group into an object like: {Pro: [...], Free: [...]}.

**Test Cases**:

* **Input**: [{name: "Alice", type: "Pro"}, {name: "Bob", type: "Free"}, {name: "Charlie", type: "Pro"}]
  + **Output**: {Pro: ["Alice", "Charlie"], Free: ["Bob"]}
* **Input**: [{name: "Dave", type: "Free"}]
  + **Output**: {Pro: [], Free: ["Dave"]}
* **Input**: []
  + **Output**: {Pro: [], Free: []}

**3️⃣ Find users with no orders**

**Briefing**: Given users (with userId) and orders (with userId), return an array of userIds who have no orders.

**Test Cases**:

* **Input**: users: [{userId: 1}, {userId: 2}], orders: [{userId: 1}]
  + **Output**: [2]
* **Input**: users: [{userId: 1}, {userId: 2}], orders: [{userId: 1}, {userId: 2}]
  + **Output**: []
* **Input**: users: []
  + **Output**: []

**4️⃣ Remove duplicates by SKU**

**Briefing**: Given products with sku, return a unique list keeping only the first occurrence.

**Test Cases**:

* **Input**: [{sku: "A1"}, {sku: "A1"}, {sku: "A2"}]
  + **Output**: [{sku: "A1"}, {sku: "A2"}]
* **Input**: [{sku: "B1"}, {sku: "B2"}]
  + **Output**: [{sku: "B1"}, {sku: "B2"}]
* **Input**: []
  + **Output**: []

**5️⃣ Top 3 selling products**

**Briefing**: Given sales = [{productId, qty}], return the top 3 productIds by total quantity sold.

**Test Cases**:

* **Input**: [{productId:1, qty:10}, {productId:2, qty:15}, {productId:3, qty:5}, {productId:2, qty:5}]
  + **Output**: [2,1,3]
* **Input**: [{productId:1, qty:5}]
  + **Output**: [1]
* **Input**: []
  + **Output**: []

**6️⃣ Find products missing price**

**Briefing**: Return products where price is null or undefined.

**Test Cases**:

* **Input**: [{name:"Laptop",price:1000},{name:"Phone",price:null},{name:"Tablet"}]
  + **Output**: [{name:"Phone",price:null},{name:"Tablet"}]
* **Input**: [{name:"Book",price:20}]
  + **Output**: []
* **Input**: []
  + **Output**: []

**7️⃣ Build price range summary**

**Briefing**: Group products into {low: <1000, mid: 1000–3000, high: >3000}.

**Test Cases**:

* **Input**: [{name:"Pen",price:500},{name:"Laptop",price:2000},{name:"Car",price:5000}]
  + **Output**: {low:[{name:"Pen",price:500}], mid:[{name:"Laptop",price:2000}], high:[{name:"Car",price:5000}]}
* **Input**: [{name:"Book",price:999}]
  + **Output**: {low:[{name:"Book",price:999}], mid:[], high:[]}
* **Input**: []
  + **Output**: {low:[], mid:[], high:[]}

**8️⃣ Merge product lists, remove duplicates by sku**

**Briefing**: Merge list1 and list2, removing duplicates by sku (keep first occurrence).

**Test Cases**:

* **Input**: list1:[{sku:"A1"},{sku:"A2"}], list2:[{sku:"A2"},{sku:"A3"}]
  + **Output**: [{sku:"A1"},{sku:"A2"},{sku:"A3"}]
* **Input**: list1: [], list2:[{sku:"B1"}]
  + **Output**: [{sku:"B1"}]
* **Input**: list1:[], list2:[]
  + **Output**: []

**9️⃣ Group orders by date**

**Briefing**: Group orders into an object: {date: [orders...]}.

**Test Cases**:

* **Input**: [{id:1,date:"2025-07-01"},{id:2,date:"2025-07-01"},{id:3,date:"2025-07-02"}]
  + **Output**: {"2025-07-01":[{...},{...}], "2025-07-02":[{...}]}
* **Input**: [{id:1,date:"2025-07-01"}]
  + **Output**: {"2025-07-01":[{id:1,date:"2025-07-01"}]}
* **Input**: []
  + **Output**: {}

**🔟 Count products per category**

**Briefing**: Return an object {category: count}.

**Test Cases**:

* **Input**: [{category:"Electronics"},{category:"Books"},{category:"Electronics"}]
  + **Output**: {Electronics:2, Books:1}
* **Input**: [{category:"Clothing"}]
  + **Output**: {Clothing:1}
* **Input**: []
  + **Output**: {}

**1️⃣1️⃣ Find product combinations ≈ budget**

**Briefing**: Find product pairs whose combined price ≈ budget (±10%).

**Test Cases**:

* **Input**: products:[{price:500},{price:1500},{price:2000}], budget:2000
  + **Output**: [[500,1500]]
* **Input**: budget:3000
  + **Output**: []
* **Input**: products: []
  + **Output**: []

**1️⃣2️⃣ Filter products by keyword**

**Briefing**: Return products where name or category includes the keyword (case-insensitive).

**Test Cases**:

* **Input**: keyword:"shoe", products:[...]
  + **Output**: [...]
* **Input**: keyword:"electronics", products:[{category:"Electronics"}]
  + **Output**: [...]
* **Input**: keyword:"hat", products:[]
  + **Output**: []

**1️⃣3️⃣ Group users by country**

**Briefing**: Group users into an object {country: [users...]}.

**Test Cases**:

* **Input**: [{country:"USA"},{country:"Canada"},{country:"USA"}]
  + **Output**: {USA:[...], Canada:[...]}
* **Input**: []
  + **Output**: {}

**1️⃣4️⃣ Find products never sold**

**Briefing**: Return products whose id never appeared in sales.

**Test Cases**:

* **Input**: products:[{id:1},{id:2}], sales:[{productId:1}]
  + **Output**: [{id:2}]
* **Input**: products:[]
  + **Output**: []

**1️⃣5️⃣ Detect duplicate orders (same user, date, total)**

**Briefing**: Find duplicate orders by same userId, date, and total.

**Test Cases**:

* **Input**: [{userId:1,date:"2025-07-01",total:100},...]
  + **Output**: [{...},{...}]
* **Input**: []
  + **Output**: []

**1️⃣6️⃣ Compute product stock value**

**Briefing**: Compute total value: sum of price \* stock for all products.

**Test Cases**:

* **Input**: [{price:1000,stock:2},{price:500,stock:3}]
  + **Output**: 3500
* **Input**: []
  + **Output**: 0

**1️⃣7️⃣ Top buyer per month**

**Briefing**: Return {month: userId} of the top buyer per month (by total).

**Test Cases**:

* **Input**: [{userId:1,date:"2025-07",total:1000},{userId:2,date:"2025-07",total:500}]
  + **Output**: {"2025-07":1}
* **Input**: []
  + **Output**: {}

**1️⃣8️⃣ Products added last 7 days (assume today: 2025-07-31)**

**Briefing**: Return products where createdAt is within last 7 days.

**Test Cases**:

* **Input**: [{createdAt:"2025-07-30"},{createdAt:"2025-07-20"}]
  + **Output**: [{createdAt:"2025-07-30"}]
* **Input**: []
  + **Output**: []

**1️⃣9️⃣ Flatten nested categories**

**Briefing**: Return a flat list of category names from nested children.

**Test Cases**:

* **Input**: [{name:"Electronics",children:[{name:"Phones"}]},{name:"Books"}]
  + **Output**: ["Electronics","Phones","Books"]
* **Input**: []
  + **Output**: []

**2️⃣0️⃣ Highest revenue day**

**Briefing**: Find date with highest total revenue.

**Test Cases**:

* **Input**: [{date:"2025-07-01",total:1000},{date:"2025-07-01",total:500},{date:"2025-07-02",total:1500}]
  + **Output**: "2025-07-01" or "2025-07-02"
* **Input**: []
  + **Output**: "" or null

**2️⃣1️⃣ Products low on stock (<10)**

**Briefing**: Given an array of product objects with stock, return an array of products where the stock is less than 10.

**Test Cases**:

* **Input**: products = [{name: "Laptop", stock: 5}, {name: "Phone", stock: 15}, {name: "Pen", stock: 0}]
  + **Output**: [{name: "Laptop", stock: 5}, {name: "Pen", stock: 0}]
* **Input**: products = [{name: "Book", stock: 10}]
  + **Output**: []
* **Input**: products = []
  + **Output**: []

**2️⃣2️⃣ Average product rating**

**Briefing**: Given an array of ratings (numbers), compute the average rating and round it to one decimal place. Return a number.

**Test Cases**:

* **Input**: ratings = [4, 5, 3, 4]
  + **Output**: 4.0
* **Input**: ratings = [3.7, 4.2, 5]
  + **Output**: 4.3
* **Input**: ratings = []
  + **Output**: 0.0

**2️⃣3️⃣ Find longest product name**

**Briefing**: Given an array of product objects with name, return the name of the product with the longest string length. If tied, return any.

**Test Cases**:

* **Input**: products = [{name: "Laptop"}, {name: "Smartphone"}, {name: "Pen"}]
  + **Output**: "Smartphone"
* **Input**: products = [{name: "Book"}, {name: "Note"}]
  + **Output**: "Book" (or "Note")
* **Input**: products = []
  + **Output**: ""

**2️⃣4️⃣ Products with price gaps (>50% diff to next)**

**Briefing**: Given an array of product objects with price, sort by price and find products where price difference to next product is greater than 50% of current price.

**Test Cases**:

* **Input**: products = [{name: "Pen", price: 100}, {name: "Book", price: 200}, {name: "Laptop", price: 1000}]
  + **Output**: [{name: "Book", price: 200}]
* **Input**: products = [{name: "Shirt", price: 100}, {name: "Pants", price: 120}]
  + **Output**: []
* **Input**: products = []
  + **Output**: []

**2️⃣5️⃣ Cart items above quantity 2**

**Briefing**: Given an array of cart item objects with name and qty, return an array of names for items with quantity greater than 2.

**Test Cases**:

* **Input**: cart = [{name: "Laptop", qty: 3}, {name: "Pen", qty: 1}, {name: "Book", qty: 5}]
  + **Output**: ["Laptop", "Book"]
* **Input**: cart = [{name: "Shirt", qty: 2}]
  + **Output**: []
* **Input**: cart = []
  + **Output**: []

**⚙️ Higher-Order Functions & Utilities (Questions 26–30)**

**2️⃣6️⃣ Build pluck(list, key)**

**Briefing**: Implement a pluck function to extract values for a given key from an array of objects.

**Test Cases**:

* **Input**: list = [{name: "Laptop", price: 1000}, {name: "Phone", price: 500}], key = "price"
  + **Output**: [1000, 500]
* **Input**: list = [{id: 1, name: "Pen"}, {id: 2, name: "Book"}], key = "name"
  + **Output**: ["Pen", "Book"]
* **Input**: list = [], key = "price"
  + **Output**: []

**2️⃣7️⃣ Build custom filter**

**Briefing**: Implement a filter function that takes an array and predicate function, returning elements passing the predicate.

**Test Cases**:

* **Input**: list = [1, 2, 3, 4], predicate = x => x % 2 === 0
  + **Output**: [2, 4]
* **Input**: list = [{price: 100}, {price: 200}], predicate = obj => obj.price > 150
  + **Output**: [{price: 200}]
* **Input**: list = [], predicate = x => true
  + **Output**: []

**2️⃣8️⃣ once(fn)**

**Briefing**: Implement once so the given function can only be executed once. Subsequent calls return undefined.

**Test Cases**:

* **Input**: fn = () => "clicked", onceFn = once(fn), calls: onceFn(), onceFn()
  + **Output**: "clicked", undefined
* **Input**: fn = x => x \* 2, onceFn = once(fn), calls: onceFn(5), onceFn(10)
  + **Output**: 10, undefined
* **Input**: fn = () => {}, onceFn = once(fn), calls: onceFn()
  + **Output**: undefined

**2️⃣9️⃣ Compose two functions**

**Briefing**: Implement compose(f, g) to produce f(g(x)).

**Test Cases**:

* **Input**: f = x => x + 1, g = x => x \* 2, compose(f, g)(5)
  + **Output**: 11
* **Input**: f = x => x.toUpperCase(), g = x => x + "a", compose(f, g)("test")
  + **Output**: "TESTA"
* **Input**: f = x => x, g = x => x, compose(f, g)(0)
  + **Output**: 0

**3️⃣0️⃣ Memoize tax calculation**

**Briefing**: Memoize a function to calculate tax as price \* taxRate to cache results for identical inputs.

**Test Cases**:

* **Input**: taxFn = memoize(price => price \* 0.1), calls: taxFn(100), taxFn(100), taxFn(200)
  + **Output**: 10, 10, 20
* **Input**: taxFn = memoize(price => price \* 0.1), calls: taxFn(50), taxFn(50)
  + **Output**: 5, 5
* **Input**: taxFn = memoize(price => price \* 0.1), calls: taxFn(0)
  + **Output**: 0

**🧰 String & Date Utilities (Questions 36–40)**

**3️⃣6️⃣ Mask credit card**

**Briefing**: Mask all but the last 4 digits of a 16-digit card number with \*.

**Test Cases**:

* **Input**: "1234567890123456"
  + **Output**: "\*\*\*\*\*\*\*\*\*\*\*\*3456"
* **Input**: "1111222233334444"
  + **Output**: "\*\*\*\*\*\*\*\*\*\*\*\*4444"
* **Input**: ""
  + **Output**: ""

**3️⃣7️⃣ Round price to 2 decimals**

**Briefing**: Round a price number to exactly 2 decimal places.

**Test Cases**:

* **Input**: 1299.567
  + **Output**: 1299.57
* **Input**: 10.1
  + **Output**: 10.10
* **Input**: 0
  + **Output**: 0.00

**3️⃣8️⃣ Format price: ₹1,299.00**

**Briefing**: Format price with ₹ symbol, commas, and two decimal places.

**Test Cases**:

* **Input**: 1299
  + **Output**: "₹1,299.00"
* **Input**: 50.5
  + **Output**: "₹50.50"
* **Input**: 0
  + **Output**: "₹0.00"

**3️⃣9️⃣ Generate random order ID**

**Briefing**: Return ID as "ORD-XXXXXX" with 6-digit random number.

**Test Cases**:

* **Input**: None
  + **Output**: e.g., "ORD-123456"
* **Input**: None (multiple calls)
  + **Output**: ["ORD-987654", "ORD-456789"]
* **Input**: None (regex check)
  + **Output**: Matches /^ORD-\d{6}$/

**4️⃣0️⃣ Validate password (≥8, 1 upper, 1 digit, 1 special)**

**Briefing**: Validate password meets rules: ≥8 chars, 1 uppercase, 1 digit, 1 special.

**Test Cases**:

* **Input**: "Abcd@1234"
  + **Output**: true
* **Input**: "abcd1234"
  + **Output**: false
* **Input**: "A@bc"
  + **Output**: false

**🔄 Async, Promises & Fetch (Questions 61–70)**

**6️⃣1️⃣ Fetch product & reviews, merge**

**Briefing**:  
Fetch product and its reviews from two separate APIs (using product ID) and merge into a single object:

{ product: {...}, reviews: [...] }

**Test Cases**:

* **Input**: Product API returns {id: 1, name: "Laptop"}, Reviews API returns [{rating: 5}, {rating: 4}]  
  **Output**: { product: {id: 1, name: "Laptop"}, reviews: [{rating: 5}, {rating: 4}] }
* **Input**: Product API returns {id: 2}, Reviews API returns []  
  **Output**: { product: {id: 2}, reviews: [] }
* **Input**: Product API fails with "Not found"  
  **Output**: Throws "Not found"

**6️⃣2️⃣ Cancel fetch on unmount**

**Briefing**:  
Use AbortController to cancel a fetch when a component unmounts; make sure no state updates happen after abort.

**Test Cases**:

* **Input**: fetchFn = () => Promise.resolve("data"), fetch completes, no abort  
  **Output**: "data"
* **Input**: fetchFn = () => new Promise(resolve => setTimeout(() => resolve("data"), 1000)), abort after 500ms  
  **Output**: Throws DOMException: Aborted
* **Input**: fetchFn = () => Promise.reject("error")  
  **Output**: Throws "error"

**6️⃣3️⃣ Poll API until complete**

**Briefing**:  
Poll an API every 5 seconds until it returns {status: "complete"}. Return the final response, or throw an error if it fails.

**Test Cases**:

* **Input**: API returns {status: "pending"} twice, then {status: "complete", data: "done"}  
  **Output**: {status: "complete", data: "done"}
* **Input**: API returns {status: "pending"}, then fails with "error"  
  **Output**: Throws "error"
* **Input**: API returns {status: "complete"} immediately  
  **Output**: {status: "complete"}

**6️⃣4️⃣ Always‑resolve wrapper**

**Briefing**:  
Create a wrapper so any promise always resolves:

* success ➜ { success: true, data: ... }
* failure ➜ { success: false, error: ... }

**Test Cases**:

* **Input**: Promise.resolve("data")  
  **Output**: { success: true, data: "data" }
* **Input**: Promise.reject("error")  
  **Output**: { success: false, error: "error" }
* **Input**: Promise.resolve(null)  
  **Output**: { success: true, data: null }

**6️⃣5️⃣ Chain: get token → user → orders**

**Briefing**:  
Chain three sequential requests:

1. get token → 2) fetch user with token → 3) fetch orders using user ID. Return the orders.

**Test Cases**:

* **Input**: Token API → {token: "abc"}, User API → {id: 1}, Orders API → [{item: "Laptop"}]  
  **Output**: [{item: "Laptop"}]
* **Input**: Token API fails with "Unauthorized"  
  **Output**: Throws "Unauthorized"
* **Input**: Token API → {token: "xyz"}, User API → {id: 2}, Orders API → []  
  **Output**: []

**6️⃣6️⃣ Refresh token if expired**

**Briefing**:  
If a request fails with "Token expired", refresh token, retry original request; return result.

**Test Cases**:

* **Input**: Original fetch fails "Token expired", refresh → "new-token", retry → "data"  
  **Output**: "data"
* **Input**: Original fetch → "success"  
  **Output**: "success"
* **Input**: Original fetch fails "Token expired", refresh also fails  
  **Output**: Throws refresh error

**6️⃣7️⃣ Limit concurrency to 3 fetches**

**Briefing**:  
Limit to at most 3 concurrent fetches; queue others until slots free.

**Test Cases**:

* **Input**: 5 fetches each return "data1"–"data5"  
  **Output**: ["data1","data2","data3","data4","data5"] (max 3 concurrent)
* **Input**: 2 fetches → "a","b"  
  **Output**: ["a","b"]
* **Input**: 4 fetches, one fails with "error"  
  **Output**: Throws "error" (still max 3 concurrent)

**6️⃣8️⃣ Batch fetch pages sequentially**

**Briefing**:  
Fetch multiple API pages (e.g., page 1 then page 2) **one after another**; combine into one array.

**Test Cases**:

* **Input**: Page 1 → ["a","b"], Page 2 → ["c"]  
  **Output**: ["a","b","c"]
* **Input**: Page 1 → [], stop there  
  **Output**: []
* **Input**: Page 1 fails "error"  
  **Output**: Throws "error"

**6️⃣9️⃣ WebSocket: listen to updates**

**Briefing**:  
Use WebSocket to listen for real‑time updates; on message, call callback with data.

**Test Cases**:

* **Input**: Server sends {type: "update", data: "new"}  
  **Output**: Callback called with {type: "update", data: "new"}
* **Input**: No messages sent  
  **Output**: Callback never called
* **Input**: WebSocket errors  
  **Output**: Callback not called; error logged/thrown

**7️⃣0️⃣ Delay promise**

**Briefing**:  
Return promise that resolves to given message after specified delay (ms).

**Test Cases**:

* **Input**: "hello", 1000  
  **Output**: Resolves to "hello" after 1000ms
* **Input**: "", 0  
  **Output**: Resolves to "" immediately
* **Input**: "test", 500  
  **Output**: Resolves to "test" after 500ms