```
// Clear the console before running this script for cleaner output

console.clear();

/*
======================================================================

                Object Interview Questions 🚀

======================================================================

*/

/*

1. Compare two objects for equality in terms of properties and values.

*/

// Define first object with 3 properties

let obj1 = { name: "Vinod", age: 30, isStudent: false };

// Define second object with 4 properties (extra 'country')

let obj2 = { name: "Vinod", age: 30, isStudent: false, country: "India" };

// Function to compare two objects

const compareObjects = (obj1, obj2) => {

    // Compare the number of keys in both objects

    if (Object.keys(obj1).length !== Object.keys(obj2).length) {

        return false; // If not equal in length, they are not equal
    }
```

```
38
39      // Loop through each key in obj1
40
41      for (let key in obj1) {
42
43          // Check if the value for the current key is not equal in both objects
44
45          if (obj1[key] !== obj2[key]) {
46
47              return false; // If mismatch found, return false
48          }
49      }
50
51      // If all keys matched, return true
52
53      return true;
54  }
55
56  // Test the function
57
58  console.log(compareObjects(obj1, obj2)); // false
59
60  /*
61
62  2. Add a new subject & grade to a student's record.
63
64  */
65
66  // Student object with nested grades object
67
68  let studentData = {
69
70      name: "Bobby",
71
72      age: 21,
73
74      grades: {
75
```

```
        Maths: 90,

        Science: 85,

        History: 88
    }
}

// Function to add a subject and marks to the student object

const addSubject = (student, subject, marks) => {

    // Check if 'grades' property exists

    if (!student.grades) {

        student.grades = {}; // If not, create it
    }

    // Add the subject and marks to grades

    return (student.grades[subject] = marks);
}

// Add "English" subject with marks 95

addSubject(studentData, "English", 95);

// Print updated student data

console.log(studentData);

// Employee object with nested personalInfo object

let employeeData = {

    name: "John",

```

```
    age: 30,

    personalInfo: {

        address: "123 Main St",

        phone: "555-555-5555"
    }
}

// Function to add email property inside personalInfo

const addEmail = (employee, email) => {

    return employee.personalInfo.email = email; // Assign new email property
}

// Add email to employeeData

addEmail(employeeData, "i8o9g@example.com");

console.log(employeeData); // Check updated object

// Function to delete email from personalInfo

const deleteEmail = (employee) => {

    delete employee.personalInfo.email; // Remove email property
}

// Delete email from employeeData

deleteEmail(employeeData);

console.log(employeeData); // Check updated object

/*

```

```
3. Shallow clone an object using spread operator.

*/

// Function to clone object (shallow copy)

const cloneObject = (obj) => {

    return { ...obj }; // Spread operator copies only first-level properties
}

// Original object with nested address object

let originalObject = {

    name: "Alice",

    age: 25,

    address: {

        city: "New York",

        state: "NY"
    }
}

// Clone originalObject

let clone = cloneObject(originalObject);

console.log(clone); // Shows copied object

// Modify nested property in cloned object

clone.address.city = "San Francisco";

// Since it's a shallow copy, original object's nested object is also modified
```

```
190
191 console.log(clone); // Shows updated city
192
193 console.log(originalObject); // Also shows updated city
194
195 /*
196
197 4. Merge two objects. Second object's keys overwrite first's if same.
198
199 */
200
201 // Function to merge two objects
202
203 const mergeObjects = (obj1, obj2) => {
204
205     return { ...obj1, ...obj2 }; // Spread both objects, second overwrites
206 }
207
208 // First object
209
210 const firstObj = { x: 1, y: 2 };
211
212 // Second object (y overwrites)
213
214 const secondObj = { y: 3, z: 4 };
215
216 // Print merged result
217
218 console.log(mergeObjects(firstObj, secondObj));
219
220 /*
221
222 5. Count number of properties in an object.
223
224 */
225
226 // Function to count number of keys
227
```

```
228 const countProperties = (obj) => {
229
230     return Object.keys(obj).length; // Object.keys returns array of keys
231 }
232
233 // Print number of keys in example object
234
235 console.log("The number of properties (keys) in the object is ", countProperties({ x: 1, y: 2, z: 3 }));
236
237 /*
238
239 6. Check if a property exists in an object.
240
241 */
242
243 // Function to check property existence
244 const hasProperty = (obj, key) => {
245
246     return obj.hasOwnProperty(key); // Returns true if property exists directly on object
247 }
248
249 // Example object
250
251 let user = {
252
253     id: 1,
254
255     username: "John Doe"
256 }
257
258 console.log(hasProperty(user, "id")); // true
259
260 console.log(hasProperty(user, "email")); // false
261
262 /*
263
264 7. Convert object to array of key-value pairs.
265
```

```
*/

// Function to get array of [key, value] pairs

const ObjectToPairs = (obj) => {

    return Object.entries(obj); // Returns array of arrays
}

console.log(ObjectToPairs({ a: 1, b: 2, c: 3 }));

/*

8. Remove a specific key from an object.

*/

// Function to remove key

const removeKey = (obj, key) => {

    delete obj[key]; // Deletes the key from object

    return obj; // Return updated object
}

// Example

let item = { id: 1, name: "IPhone", price: 100000 };

console.log(removeKey(item, "id"));

/*

9. Iterate over all keys and values in an object.

*/
```

```
// Function to print key-value pairs
const printObject = (obj) => {

    for (let [key, value] of Object.entries(obj)) { // Destructure [key, value]

        console.log(`${key}: ${value}`); // Print in key: value format
    }
}

// Example object

let userProfile = { name: "Sara", Profession: "Software Engineer" };

printObject(userProfile);

/*

10. Get only keys or values from an object.

*/

// Example object

let sampleObj = { a: 10, b: 20, c: 30 };

// Get all keys

console.log(Object.keys(sampleObj));

// Get all values

console.log(Object.values(sampleObj));

/*

11. Convert an object to a JSON string.

*/
```

```
342
343 // Function to stringify object
344
345 const objectToString = (obj) => {
346
347     return JSON.stringify(obj); // Converts to JSON string
348 }
349
350 console.log(objectToString({ name: "John", age: 25, city: "New York" }));
351
352 /*
353
354 12. Convert a JSON string back to an object.
355
356 */
357
358 // Function to parse JSON string
359
360 const stringToObject = (str) => {
361
362     return JSON.parse(str); // Converts string to object
363 }
364
365 console.log(stringToObject('{"name": "John", "age": 25, "city": "New York"}'));
366
367 /*
368
369 13. Check if an object is empty.
370
371 */
372
373 // Function to check if object has zero keys
374
375 const isEmptyObject = (obj) => {
376
377     return Object.keys(obj).length === 0;
378 }
379
```

```
console.log(isEmptyObject({})); // true

console.log(isEmptyObject({ name: "John", age: 25 })); // false

/*

14. Get the first key in an object.

*/

// Function to get first key

const getFirstKey = (obj) => {

    return Object.keys(obj)[0];
}

console.log(getFirstKey({ a: 1, b: 2, c: 3 }));

/*

15. Get the last key in an object.

*/

// Function to get last key

const getLastKey = (obj) => {

    return Object.keys(obj)[Object.keys(obj).length - 1];
}

console.log(getLastKey({ a: 11, b: 22, c: 33 }));

/*

16. Get the first value in an object.

```

```
*/

// Function to get first value

const getFirstValue = (obj) => {

    return Object.values(obj)[0];
}

console.log(getFirstValue({ a: 100, b: 200, c: 300 }));

/*

17. Get the last value in an object.

*/

// Function to get last value

const getLastValue = (obj) => {

    return Object.values(obj)[Object.values(obj).length - 1];
}

console.log(getLastValue({ a: 1000, b: 2000, c: 3000 }));

/*

18. Get the first key-value pair in an object.

*/

// Function to get first entry

const getFirstKeyValuePair = (obj) => {

    return Object.entries(obj)[0];
}
```

```
456
457 console.log(getFirstKeyValuePair({ a: 1, b: 2, c: 3 }));
458
459 /*
460
461 19. Get the last key-value pair in an object.
462
463 */
464
465 // Function to get last entry
466
467 const getLastKeyValuePair = (obj) => {
468
469     return Object.entries(obj)[Object.entries(obj).length - 1];
470 }
471
472 console.log(getLastKeyValuePair({ a: 10000, b: 20000, c: 30000 }));
473
474 /*
475
476 20. Get sum of all values in an object.
477
478 */
479
480 // Function to sum values
481
482 const getSumOfValues = (obj) => {
483
484     return Object.values(obj).reduce((acc, currentValue) => acc + currentValue, 0);
485 }
486
487 console.log(getSumOfValues({ a: 10, b: 20, c: 30 }));
488
489 /*
490
491 21. Get average of all values in an object.
492
493 */
```

```
494
495 // Function to average values
496
497 const getAverageOfValues = (obj) => {
498
499     return Object.values(obj).reduce((acc, currentValue) => acc + currentValue, 0) / Object.values(obj).length;
500 }
501
502 console.log(getAverageOfValues({ a: 10, b: 20, c: 30 }));
503
504 /*
505
506 22. Get maximum value in an object.
507
508 */
509
510 // Function to find max value
511
512 const getMaxValue = (obj) => {
513
514     return Math.max(...Object.values(obj));
515 }
516
517 console.log(getMaxValue({ a: 10, b: 20, c: 30 }));
518
519 /*
520
521 23. Get minimum value in an object.
522
523 */
524
525 // Function to find min value
526
527 const getMinValue = (obj) => {
528
529     return Math.min(...Object.values(obj));
530 }
531
```

```
console.log(getMinValue({ a: 10, b: 20, c: 30 }));

/*

24. Get length of the longest key in an object.

*/

// Function to find length of longest key

const getLongestKey = (obj) => {

    return Math.max(...Object.keys(obj).map((key) => key.length));
}

console.log(getLongestKey({ a: 10, b: 20, c: 30 }));

/*

25. Get length of the shortest key in an object.

*/

// Function to find length of shortest key

const getShortestKey = (obj) => {

    return Math.min(...Object.keys(obj).map((key) => key.length));
}

console.log(getShortestKey({ a: 10, b: 20, c: 30 }));
```