JAVASCRIPT DAY 20th

CHEATSHEET

1. JSON Basics

- **Definition:** JSON (JavaScript Object Notation) is a lightweight format for data exchange.

Features:

- Human-readable and machine-parseable.
- Uses key-value pairs, arrays, and objects for structuring data.
- Widely used for communication between servers and web applications.

2. JSON Syntax

- **Objects:** Represented by { }, containing key-value pairs.
- **Arrays:** Represented by [ ], containing multiple values.

Key-Value Structure:

- Keys and values are separated by :.
- Pairs are separated by ,.

3. Extracting Data

Access Methods:

1. **Dot (.) notation:** E.g., object.key
2. **Square bracket ([]) notation:** E.g., object["key"]
3. **Destructuring:** Simplifies variable assignment.

Destructuring Types:

- **Object Destructuring:** Extracts properties from objects.
- **Array Destructuring:** Extracts elements from arrays.
- **Nested Destructuring:** Handles deeply nested structures.

4. JSON Operations

Concatenation: Merge JSON objects using:

- Spread operator: ...
- Object.assign().

Accessing Keys and Values:

- Object.keys() for keys.
- Object.values() for values.

Looping:

- Use for...in loops or Object.entries() for iteration.

5. Working with Strings

- **Convert Object to String:**
- JSON.stringify() converts JSON objects into strings.

Convert String to Object:

- JSON.parse() parses JSON strings into objects.

6. JSON Mutability

JSON objects are mutable, allowing property:

- Addition.
- Updates.
- Deletion.

Examples:

- delete operator.
- Object.defineProperty() and Object.defineProperties() for advanced control.

7. JSON Constraints

- **Duplicate Keys:**
- Not allowed. Only the last value for a key is retained.

Duplicate Values:

- Allowed without overwriting each other.

8. Advanced JSON Operations

- **Object.freeze():** Makes objects immutable, preventing modifications.
- **Object.seal():** Prevents adding/removing properties but allows updates.
- **Object.entries():** Converts an object into an array of key-value pairs.
- **Object.fromEntries():** Converts an array of key-value pairs back into an object.

9. Practical Examples

- **Displaying Data in Tables:** Iterate over arrays of objects (e.g., employee data) and display in HTML tables.
- **Nested Object Handling:** Use nested destructuring to easily access deeply nested properties.

10. JSON Utilities

- **hasOwnProperty():** Checks if a specific key exists as a direct property of an object.
- **Default Values:** Provide default values for undefined properties during destructuring.

CODING QUESTIONS OF DAY 20

1. JSON Basics

1. **Write a JSON object to store the details of a student with the fields: name, age, grade, and subjects (as an array).**

```
let student = {
  name: "John Doe",
  age: 20,
  grade: "A",
  subjects: ["Math", "Science", "History"]
};
console.log(student);
```

2. **Create a JSON object to represent a product with attributes like product ID, name, price, and availability (true/false).**

```
let product = {
  productID: 101,
  name: "Laptop",
  price: 1200.50,
  availability: true
};
console.log(product);
```

2. Accessing Data

3. **Create a JSON object for a person with nested objects for "address" and "contact". Access the city from the "address" and the phone number from the "contact" using dot notation.**

```
let person = {
  name: "Alice",
  address: {
    city: "New York",
    zip: "10001"
  },
  contact: {
    phone: "123-456-7890",
```

```
    email: "alice@example.com"
  }
};
console.log(person.address.city);  // Access city
console.log(person.contact.phone);  // Access phone number
```

4. **Write a program to access and display all keys of a given JSON object using Object.keys().**

```
let person = {
  name: "Alice",
  age: 30,
  city: "New York"
};
console.log(Object.keys(person));  // Output: ['name', 'age', 'city']
```

5. **Extract and display the values of a JSON object using Object.values().**

```
let person = {
  name: "Alice",
  age: 30,
  city: "New York"
};
console.log(Object.values(person));  // Output: ['Alice', 30, 'New York']
```

3. Destructuring

6. **Create an object with keys: frontend, backend, and database. Use destructuring to assign these values to individual variables and display them.**

```
let skills = {
  frontend: "React",
  backend: "Node.js",
  database: "MongoDB"
};
let { frontend, backend, database } = skills;
console.log(frontend, backend, database);  // Output: React Node.js MongoDB
```

7. **Write a program to destructure an array [10, 20, 30, 40] into variables and print their values.**

```
let arr = [10, 20, 30, 40];
let [a, b, c, d] = arr;
```

```
console.log(a, b, c, d);  // Output: 10 20 30 40
```

8. **Use nested destructuring to extract the name of a subject from the following JSON:**

```
let obj = {
  course: {
    module: {
      subject: "JavaScript"
    }
  }
};
let { course: { module: { subject } } } = obj;
console.log(subject);  // Output: JavaScript
```

4. JSON Operations

9. **Merge two JSON objects representing "user details" and "user preferences" using the spread operator.**

```
let userDetails = { name: "Alice", age: 30 };
let userPreferences = { theme: "dark", language: "English" };

let mergedUser = { ...userDetails, ...userPreferences };
console.log(mergedUser);
```

10. **Write a program to update the value of an existing key in a JSON object.**

```
let product = {
  id: 101,
  name: "Laptop",
  price: 1200.50
};
product.price = 1100.75;  // Update the price
console.log(product);
```

11. **Use the delete operator to remove a key-value pair from a JSON object and display the updated object.**

```
let product = {
  id: 101,
  name: "Laptop",
```

```
  price: 1200.50
};
delete product.price;  // Remove the price property
console.log(product);
```

5. Conversions

12. **Convert the following JSON object into a string using JSON.stringify() and display its type:**

```
let obj = { key: "value", id: 101 };
let jsonString = JSON.stringify(obj);
console.log(jsonString);  // Output: '{"key":"value","id":101}'
console.log(typeof jsonString);  // Output: string
```

13. **Parse a JSON string into an object using JSON.parse() and display its properties.**

```
let jsonString = '{"name":"Alice","age":30}';
let parsedObject = JSON.parse(jsonString);
console.log(parsedObject.name);  // Output: Alice
console.log(parsedObject.age);   // Output: 30
```

6. Looping

14. **Write a program to iterate over a JSON object using a for...in loop and display each key-value pair.**

```
let person = { name: "Alice", age: 30, city: "New York" };
for (let key in person) {
  console.log(`${key}: ${person[key]}`);
}
```

15. **Use Object.entries() to convert a JSON object into an array of key-value pairs and iterate through it.**

```
let person = { name: "Alice", age: 30, city: "New York" };

for (let [key, value] of Object.entries(person)) {
  console.log(`${key}: ${value}`);
}
```

7. Displaying Data

16. **Create an array of JSON objects representing employees with fields: ID, name, and salary. Display this data in an HTML table.**

```
<!DOCTYPE html>
<html>
<head>
  <title>Employee Table</title>
</head>
<body>
  <table border="1">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Salary</th>
      </tr>
    </thead>
    <tbody>
      <script>
        let employees = [
          { ID: 1, name: "John", salary: 50000 },
          { ID: 2, name: "Jane", salary: 60000 },
          { ID: 3, name: "Alice", salary: 70000 }
        ];
        employees.forEach(employee => {
document.write(`<tr><td>${employee.ID}</td><td>${employee.name}</td><td>${employee.salary}</td></tr>`);
        });
      </script>
    </tbody>
  </table>
</body>
</html>
```

17. **Create an array of JSON objects representing users with fields: name and profile picture URL. Display the data in a table, showing the name and profile picture.**

```
<!DOCTYPE html>
<html>
<head>
  <title>User Table</title>
</head>
<body>
  <table border="1">
    <thead>
      <tr>
        <th>Name</th>
        <th>Profile Picture</th>
      </tr>
    </thead>
    <tbody>
      <script>
        let users = [
          { name: "Alice", profilePic: "alice.jpg" },
          { name: "Bob", profilePic: "bob.jpg" },
          { name: "Charlie", profilePic: "charlie.jpg" }
        ];

        users.forEach(user => {
          document.write(`<tr><td>${user.name}</td><td><img src="${user.profilePic}" alt="${user.name}" width="50" height="50"/></td></tr>`);
        });
      </script>
    </tbody>
  </table>
</body>
</html>
```

8. Advanced Operations

18. **Use Object.freeze() on a JSON object and attempt to modify its properties. Observe and explain the output.**

```
let obj = { name: "Alice" };
Object.freeze(obj);
obj.name = "Bob";  // This will not change the object
console.log(obj.name);  // Output: Alice
```

19. **Use Object.seal() on a JSON object, try to add and delete properties, and display the results.**

```
let obj = { name: "Alice" };
Object.seal(obj);
obj.age = 30;  // Adding a new property will not work
delete obj.name;  // Deleting an existing property will not work
console.log(obj);  // Output: { name: 'Alice' }
```

20. **Use Object.fromEntries() to convert an array of key-value pairs back into a JSON object.**

```
let entries = [["name", "Alice"], ["age", 30]];
let obj = Object.fromEntries(entries);
console.log(obj);  // Output: { name: 'Alice', age: 30 }
```

9. Real-World Scenarios

21. **Write a program to read an array of objects from an API response and display specific fields in an HTML table.**

```
<!DOCTYPE html>
<html>
<head>
  <title>API Response Table</title>
</head>
<body>
  <table border="1">
    <thead>
      <tr>
        <th>Name</th>
        <th>Email</th>
      </tr>
    </thead>
```

```
    <tbody>
      <script>
        fetch("https://jsonplaceholder.typicode.com/users")
          .then(response => response.json())
          .then(data => {
            data.forEach(user => {
              document.write(`<tr><td>${user.name}</td><td>${user.email}</td></tr>`);
            });
          });
      </script>
    </tbody>
  </table>
</body>
</html>
```

22. **Create a nested JSON object to store course details, including course name, duration, and an array of modules. Use destructuring to extract and display specific module names.**

```
let course = {
  courseName: "Web Development",
  duration: "3 months",
  modules: ["HTML", "CSS", "JavaScript"]
};
let { modules } = course;
let [module1, module2, module3] = modules;
console.log(module1, module2, module3);  // Output: HTML CSS JavaScript
```

10. Utilities and Methods

23. **Write a program to check if a key exists in a JSON object using hasOwnProperty().**

```
let person = { name: "Alice", age: 30 };
console.log(person.hasOwnProperty("name"));  // Output: true
console.log(person.hasOwnProperty("address"));  // Output: false
```

24. **Add a new key-value pair to an existing JSON object and display the updated object.**

```
let person = { name: "Alice", age: 30 };
person.address = "New York";  // Add a new property
```

```
console.log(person);
```

25. **Define multiple properties at once in a JSON object using Object.defineProperties() and display the object.**

```
let person = {};
Object.defineProperties(person, {
  name: { value: "Alice", writable: true },
  age: { value: 30, writable: true }
});
console.log(person);  // Output: { name: 'Alice', age: 30 }
```

FAQ'S OF DAY 20

1. What is JSON, and what is it used for?

Answer: JSON stands for JavaScript Object Notation. It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is primarily used to transmit data between a server and a web application or between different parts of a system in a standardized format.

2. How does JSON differ from JavaScript objects?

Answer: While JSON is similar to JavaScript objects in syntax, there are key differences:

- Keys in JSON must be strings (enclosed in double quotes), whereas in JavaScript objects, keys can be strings, symbols, or numbers.
- Values in JSON are restricted to strings, numbers, booleans, arrays, objects, and 'null', while JavaScript objects can have functions, 'undefined', or other types as values.
- JSON is purely a data format and does not include methods or any executable code, unlike JavaScript objects.

3. How do you convert a JavaScript object to a JSON string?

Answer: You can convert a JavaScript object to a JSON string using the JSON.stringify() method. Example:

```
const obj = { name: "Alice", age: 25 };
const jsonString = JSON.stringify(obj);
console.log(jsonString); // '{"name":"Alice","age":25}'
```

4. How do you convert a JSON string into a JavaScript object?

Answer: You can convert a JSON string into a JavaScript object using the JSON.parse() method. Example:

```
const jsonString = '{"name":"Alice","age":25}';
```

```
const obj = JSON.parse(jsonString);
console.log(obj); // { name: 'Alice', age: 25 }
```

5. What are the limitations of JSON?

Answer: JSON has several limitations:

- It does not support functions, ‘undefined’, or symbols as values.
- It cannot represent complex data types like dates directly (dates are usually represented as strings).
- JSON does not support circular references or other non-serializable structures.

6. How can you handle a situation where ‘JSON.parse()’ throws an error due to invalid JSON?

Answer: You can handle errors from JSON.parse() using a try...catch block to catch exceptions and deal with them gracefully.

Example:

```
try {
  const obj = JSON.parse('invalid JSON string');
} catch (error) {
  console.error("Parsing error:", error.message);
}
```

7. What is the difference between ‘JSON.stringify()’ and ‘toString()’ methods in JavaScript?

Answer:

- JSON.stringify() converts an entire JavaScript object or array to a JSON string. It recursively serializes the object, including nested objects or arrays.
- toString() is a method available on many JavaScript objects and returns a string representation of that object, often in a non-JSON format (e.g., "[object Object]" for most objects).

8. How can you pretty-print a JSON string?

Answer: You can pretty-print a JSON string by using the JSON.stringify() method with optional space arguments.

Example:

```
const obj = { name: "Alice", age: 25, city: "Wonderland" };
const prettyJson = JSON.stringify(obj, null, 2);
console.log(prettyJson);
```

9. Can JSON be used to represent all JavaScript data types?

Answer: No, JSON cannot represent all JavaScript data types. JSON can represent strings,

numbers, booleans, arrays, objects, and null. It cannot represent functions, 'undefined', symbols, or other complex types like Map, Set, or Date (dates are typically represented as ISO 8601 strings).

10. What is the role of a reviver function in JSON.parse()?

Answer: The JSON.parse() method can take a second argument, a reviver function, which can transform the resulting object before it is returned. This function is called for each key-value pair in the object, allowing you to customize how values are parsed.

Example:

```
const jsonString = '{"name":"Alice","age":"25"}';
const obj = JSON.parse(jsonString, (key, value) => {
  return key === "age" ? Number(value) : value;
});
console.log(obj); // { name: 'Alice', age: 25 }
```

JSON MCQ'S

1. JSON stands for:

   A) JavaScript Object Notation

2. JSON is a format for:

   A) Storing and transporting data

3. The JSON syntax is a subset of the:

   D) JavaScript syntax

4. Who is the creator of JSON?

   C) Douglas Crockford

5. In the JSON syntax, data is separated by:

   C) Commas

6. In the JSON syntax, an array is written within:

   A) Square brackets

7. Features of JSON include:

   A) Simplicity

   B) Openness

   C) Self-Describing

   E) Extensibility

   F) Interoperability

8. The correct symbol to insert a comment in JSON is:

   D) JSON doesn't support comments

9. In the JSON syntax, data is in:

   A) Class/object name/value pairs

10. JSON names (keys) require double quotes.

    A) (True)

11. JSON names (keys) must be strings.

    A) (True)

12. File type for JSON files:

    C) .json

13. Correct syntax for writing a JSON name/value pair where the value is of string type:

    A) (True)

14. MIME type for JSON text:

    application/json

15. JSON is primarily used for:

    B) Data interchange

16. Which of these is a valid JSON value?

    B) { key: “value” }

17. In JSON, strings must be wrapped in:

    B) Double quotes (" ")

18. Which of the following data types is not supported in JSON?

    B) Functions

19. Which of the following is the correct way to represent a boolean in JSON?

    B) true

20. Which data structures can JSON represent?

    A) Objects and Arrays

21. In JSON, which of these values represents ‘no value’?

    C) null

22. How would you represent a numeric value in JSON?

    B) 10.0

23. Which symbol is used to denote the start and end of an array in JSON?

    D) [ ]

24. How would you represent a date in JSON?

    B) As a string

25. In JSON, the keys of an object must be:

    A) Strings

26. Which of the following is a valid JSON object?

A) { “name”: “John”, “age”: 30, “city”: “New York” }

27. If you had the following JSON: { “colors”: [“red”, “green”, “blue”] }, how would you describe the value associated with the key “colors”?

C) An array

28. Which of the following is not a primitive type in JSON?

D) Date

29. JSON can be used with which of the following programming languages?

D) All of the above

30. JSON is often used in conjunction with:

A) HTML

31. Which method is used to convert a JavaScript object into a JSON string?

B) JSON.stringify()

32. Which method is used to parse a JSON string into a JavaScript object?

A) JSON.parse()

33. JSON can represent:

C) Both objects and arrays

34. Which of the following is a valid JSON array?

A) [“apple”, “banana”, “cherry”]

35. JSON is commonly used for:

A) Data interchange between a server and a web application

36. JSON can be used to represent:

D) All of the above

37. JSON is more compact than XML.

True

38. JSON is easier to parse than XML.

True

39. JSON supports comments.

False

40. JSON is language-independent.

True

41. Which of the following code will throw an error?

C) JSON.parse(undefined);

42. What is JSONP meant to mitigate?

C) Cross-domain communication

43. What kind of format is JSON, and what does the acronym mean?

A) A lightweight data-interchange format. JavaScript Object Notation.

44. Which statement about the replacer parameter in JSON.stringify() is true?

D) All three statements are true

45. In this example, what is the TYPE of employee?

C) Object

46. Which of these is a proper JSON array?

B) { "letters" : ["a", "b", "c"] }

47. Which answer represents the following order of TYPES? Object, String, Boolean, Number

A) { }, "0", false, 0

48. Which statement about the reviver parameter in JSON.parse() is true?

D) All three statements are true

49. Which of these data interchange formats has seen a decline in usage in favor of JSON?

B) XML

50. Which of the following is a valid JSON string?

B) { "meals" : ["breakfast" , "lunch" , "dinner"] }