\*What will the following code output?\*\*

```javascript

const arr = [1, 2, 3, 4];

const result = arr.map((num, index, array) => array[index] \* 2);

console.log(result);

```

a) `[1, 2, 3, 4]`

b) `[2, 4, 6, 8]`

c) `[4, 8, 12, 16]`

d) `undefined`

2. \*\*What will the result of this code be?\*\*

```javascript

const arr = [10, 20, 30];

const result = arr.map((num) => num.toString());

console.log(result);

```

a) `['10', '20', '30']`

b) `[10, 20, 30]`

c) `['10', 20, '30']`

d) `undefined`

3\*\*Which of the following transforms each string to uppercase?\*\*

```javascript

const arr = ['a', 'b', 'c'];

```

a) `arr.map(str => str.toUpperCase());`

b) `arr.forEach(str => str.toUpperCase());`

c) `arr.filter(str => str.toUpperCase());`

d) `arr.map(str => str.toLowerCase());`

4. \*\*What will this code return?\*\*

```javascript

const arr = [4, 5, 6];

const result = arr.map(num => num + 1).filter(num => num > 5);

console.log(result);

```

a) `[4, 5]`

b) `[5, 6, 7]`

c) `[6, 7]`

d) `[7]`

5. \*\*What happens if `map` callback does not return a value?\*\*

```javascript

const arr = [1, 2, 3];

const result = arr.map((num) => {

console.log(num);

});

console.log(result);

```

a) `undefined`

b) `[undefined, undefined, undefined]`

c) `[null, null, null]`

d) `[]`

6. \*\*Which of the following filters numbers greater than 5?\*\*

```javascript

const arr = [1, 3, 6, 8];

```

a) `arr.map(num => num > 5);`

b) `arr.filter(num => num > 5);`

c) `arr.forEach(num => num > 5);`

d) `arr.filter(num => num < 5);`

7. \*\*What will this code output?\*\*

```javascript

const arr = [4, 5, 6];

const result = arr.filter((num, index) => index % 2 === 0);

console.log(result);

```

a) `[4]`

b) `[4, 6]`

c) `[5]`

d) `[5, 6]`

8. \*\*What will this code return?\*\*

```javascript

const arr = ['apple', 'banana', 'cherry'];

const result = arr.filter(word => word.length > 5);

console.log(result);

```

a) `['apple', 'banana']`

b) `['banana', 'cherry']`

c) `['cherry']`

d) `[]`

9. \*\*How can you filter non-null values from an array?\*\*

```javascript

const arr = [1, null, 2, undefined, 3];

```

a) `arr.filter(num => num !== null);`

b) `arr.filter(num => num != null);`

c) `arr.filter(num => !!num);`

d) All of the above

10. \*\*What does the following code output?\*\*

```javascript

const arr = [1, 2, 3];

const result = arr.filter((num, index, array) => array.includes(num + 1));

console.log(result);

```

a) `[1, 2]`

b) `[2, 3]`

c) `[1, 2, 3]`

d) `[]`

11. \*\*What is logged by the following code?\*\*

```javascript

const arr = [10, 20, 30];

arr.forEach(num => console.log(num \* 2));

```

a) Logs `10, 20, 30`

b) Logs `20, 40, 60`

c) Logs `undefined`

d) Logs `[20, 40, 60]`

\*\*What happens if you try to break out of a `forEach` loop using `break`?\*\*

a) The loop breaks immediately.

b) The loop throws an error.

c) The loop continues without interruption.

d) The loop skips the current iteration.

13. \*\*What will the following code output?\*\*

```javascript

const arr = [1, 2, 3];

const result = arr.forEach(num => num \* 2);

console.log(result);

```

a) `[2, 4, 6]`

b) `[1, 2, 3]`

c) `undefined`

d) `[]`

\*\*Can `forEach` be used to transform an array?\*\*

a) Yes, directly.

b) No, because `forEach` does not return a value.

c) Yes, but only with `return` inside the callback.

d) No, because it only works on strings.

15. \*\*How many times does the callback execute in the following code?\*\*

```javascript

const arr = [1, 2, 3];

arr.forEach((num, index) => {

if (index === 1) return;

console.log(num);

});

a) 1

b) 2

c) 3

d) None

16. \*\*What is the output of this code?\*\*

```javascript

const arr = [1, 2, 3, 4];

const result = arr.map(num => num \* 2).filter(num => num > 5);

console.log(result);

```

a) `[6, 8]`

b) `[4, 6, 8]`

c) `[2, 4, 6, 8]`

d) `[8]`

17. \*\*Which of the following code chains `map` and `forEach` correctly?\*\*

```javascript

const arr = [1, 2, 3];

```

a) `arr.map(num => num \* 2).forEach(num => num \* 3);`

b) `arr.map(num => num \* 2).forEach(num => console.log(num));`

c) `arr.forEach(num => num \* 2).map(num => console.log(num));`

d) Both b) and c)

18. \*\*What is the final result of this code?\*\*

```javascript

const arr = [1, 2, 3];

const result = arr

.map(num => num \* 2)

.filter(num => num > 2)

.forEach(num => console.log(num));

console.log(result);

```

a) `[4, 6]`

b) Logs `4, 6` and then `undefined`

c) Logs `undefined`

d) Logs `4, 6`

#### Miscellaneous

19. \*\*Which method(s) return a new array?\*\*

a) `map`

b) `filter`

c) Both a) and b)

d) None

20. \*\*What happens if the callback function passed to `map` throws an error?\*\*

a) The `map` loop stops immediately.

b) The error is caught silently.

c) The error propagates.

d) The corresponding entry in the result array is `undefined`.

21. \*\*Which of these operations are best suited for `filter`?\*\*

a) Removing duplicates

b) Summing all numbers in an array

c) Selecting specific items based on a condition

d) Transforming data

22. \*\*What does this code output?\*\*

```javascript

const arr = ['a', 'b', 'c'];

const result = arr.map((char, index) => index + char);

console.log(result);

```

a) `['0a', '1b', '2c']`

b) `['a0', 'b1', 'c2']`

c) `['a1', 'b2', 'c3']`

d) `undefined`

23. \*\*What is true about chaining `map`, `filter`, and `forEach`?\*\*

a) They all return a value that can be chained.

b) `forEach` must be used at the end since it returns `undefined`.

c) They cannot be chained together.

d) `map` must always precede `filter`.

24. \*\*What will the following code output?\*\*

```javascript

const arr = [2, 3, 4];

const result = arr

.map(num => num \*\* 2)

.filter(num => num % 2 === 0)

.map(num => num / 2);

console.log(result);

```

a) `[2, 4]`

b) `[4, 8]`

c) `[2, 8]`

d) `[1, 4, 8]`

25. \*\*Which of the following is the most efficient way to compute the sum of squares of even numbers?\*\*

```javascript

const arr = [1, 2, 3, 4, 5];

```

a)

```javascript

let sum = 0;

arr.map(num => num \*\* 2)

.filter(num => num % 2 === 0)

.forEach(num => sum += num);

```

b)

```javascript

const result = arr

.filter(num => num % 2 === 0)

.reduce((sum, num) => sum + num \*\* 2, 0);

```

c) Both a) and b) are equally efficient.

d) Neither a) nor b) is efficient.

26. \*\*What will this code output?\*\*

```javascript

const arr = [[1, 2], [3, 4]];

const result = arr.map(subArr => subArr.map(num => num \* 2));

console.log(result);

```

a) `[2, 4, 6, 8]`

b) `[[1, 2], [3, 4]]`

c) `[[2, 4], [6, 8]]`

d) `[2, 4]`

27. \*\*How do you filter an array of objects based on a specific property?\*\*

```javascript

const arr = [

{ id: 1, active: true },

{ id: 2, active: false },

{ id: 3, active: true },

];

```

a)

```javascript

arr.filter(item => item.active === true);

```

b)

```javascript

arr.filter(item => item.active);

```

c) Both a) and b).

d) Neither.

28. \*\*What is the output of this code?\*\*

```javascript

const arr = [

{ id: 1, value: 10 },

{ id: 2, value: 20 },

];

const result = arr.map(obj => ({ ...obj, value: obj.value \* 2 }));

console.log(result);

```

a)

```javascript

[{ id: 1, value: 20 }, { id: 2, value: 40 }]

```

b)

```javascript

[{ id: 1, value: 10 }, { id: 2, value: 20 }]

```

c)

```javascript

[{ id: 1, value: 2 }, { id: 2, value: 4 }]

```

d) `undefined`

29. \*\*What is the output of the following chained methods?\*\*

```javascript

const arr = [1, 2, 3, 4];

const result = arr

.map(num => ({ original: num, squared: num \*\* 2 }))

.filter(obj => obj.squared > 5);

console.log(result);

```

a)

```javascript

[{ original: 2, squared: 4 }, { original: 3, squared: 9 }]

```

b)

```javascript

[{ original: 3, squared: 9 }, { original: 4, squared: 16 }]

```

c)

```javascript

[{ original: 4, squared: 16 }]

```

d) `undefined`

30. \*\*What happens if you pass an empty array to `map`, `filter`, or `forEach`?\*\*

```javascript

const arr = [];

console.log(arr.map(num => num \* 2));

console.log(arr.filter(num => num > 2));

arr.forEach(num => console.log(num));

```

a) `[]`, `[]`, nothing logged.

b) `[0]`, `[0]`, logs `0`.

c) `undefined`, `undefined`, nothing logged.

d) Throws an error.