Table of Contents

Introduction to Object Utilities

What are Object Utilities?

Object utilities in programming, especially in JavaScript, are built-in or custom methods, functions, or techniques used to manipulate, transform, or analyze objects.
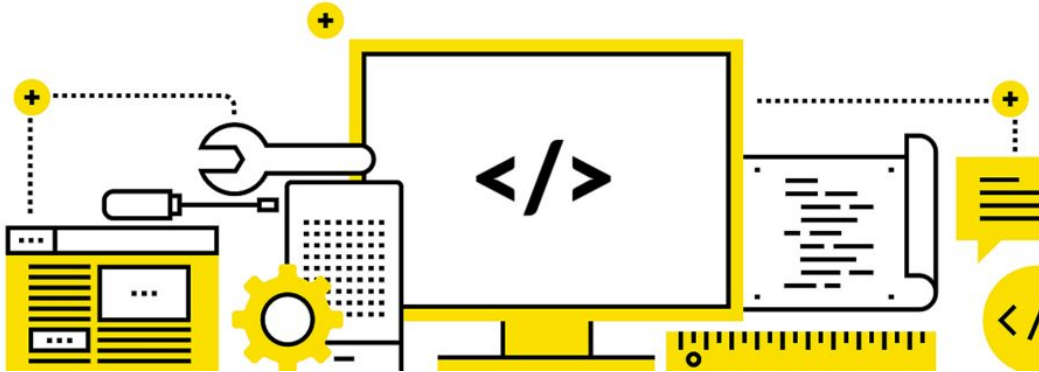
They allow to access and manipulate objects in much simpler and efficient manner.

Uses of Object Utilities

Object utilities help list properties, collect values, pair data, and rebuild structured information.

Object.keys() List properties	Object.values() Collect Values	Object.values() Pair Data	Object.fromEntries() Rebuilds information

My Collection

Collectibles
2,778

Valued Collectibles
1,893 ⓘ

Wish List
1,684

Observation #1

Observation #2

```
[
    ['Username', 'Alice'],
    ['Email', 'alice@example.com']
]

{
    username: 'Alice',
    email: 'alice@example.com'
}
```
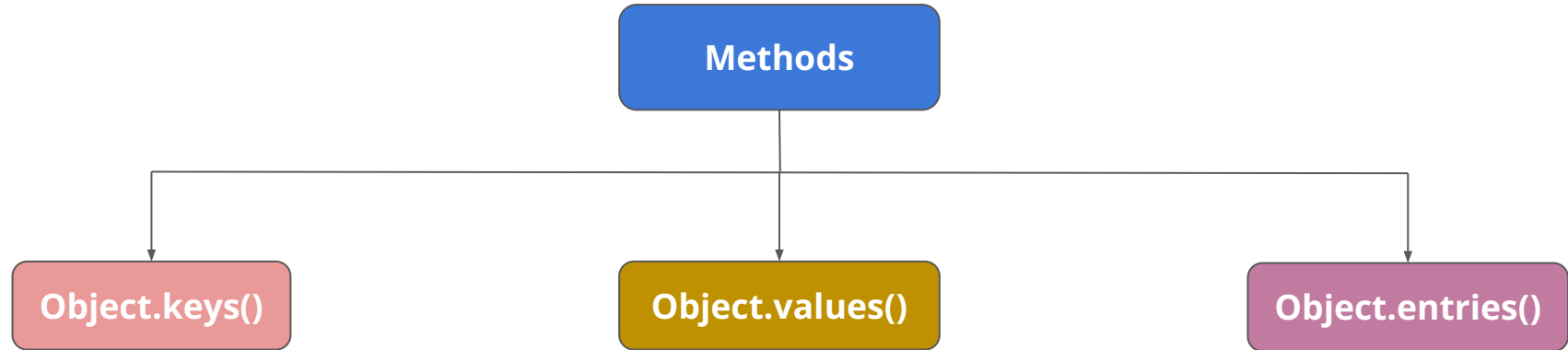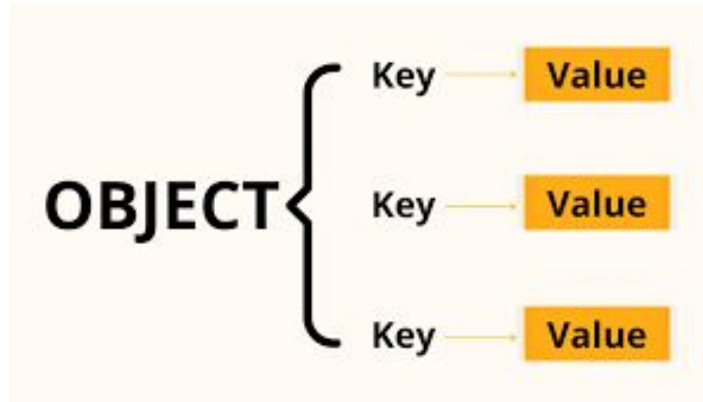
Data Retrieval Utility Methods

We have these three methods for retrieving data:-

Data Retrieval: Object.keys()

Object.keys() is a JavaScript method used to retrieve all the keys (property names) of an object as an array.

Each object has one or more key and these keys correspond to a value.

Data Retrieval: Object.keys()

Before we jump into Object.keys() we need to understand the structure of an object.

Here there are two properties each having a key and value.


```
                                            ──────────── Object name
          let user = {
            name: "John Doe", ◄──── property
  Object ◄─  age:  40 ◄──────────────── value
          }
                                        ──────────── key
```
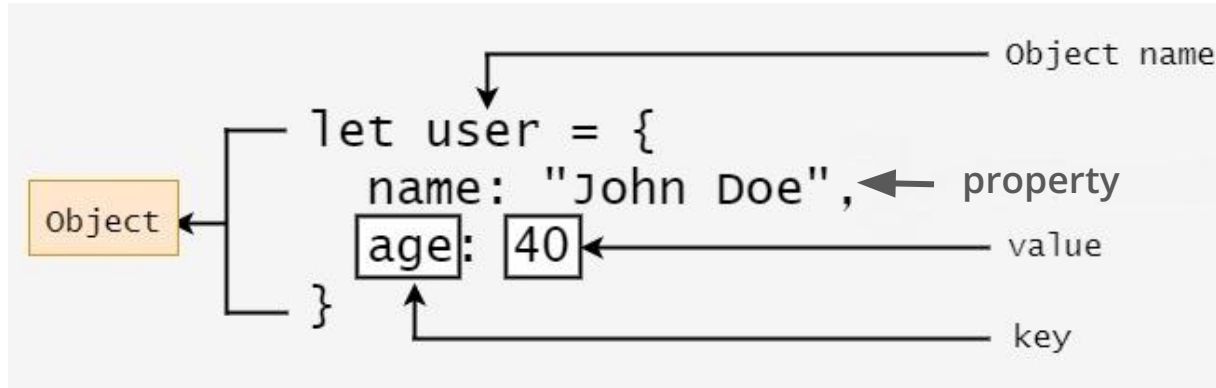
We can retrieve values of an property using keys in that object.

Data Retrieval: Object.keys()

We can retrieve the values using keys, but how to retrieve keys?? Object.keys() method comes for rescue. Here is an example:-

```javascript
1   const person = {
2       name: 'Alice',
3       age: 25,
4       city: 'New York'
5   };
6
7   const keys = Object.keys(person);
8
9   console.log(keys);
10  // Output: ['name', 'age', 'city']
```
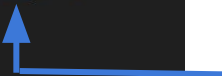
Object.keys() method takes an object as an argument and returns all the keys in form of an array.

Object.keys(): Use Cases

We can use Object.keys() to get an array of keys and then can iterate over that array to access the values of that object. Here is an example:-

```
1  Object.keys(person).forEach(key => {
2      console.log(`${key}: ${person[key]}`);
3  });
4  // Output:
5  // name: Alice
6  // age: 25
7  // city: New York
```

Accessing the value in the object *person* using its keys.

Object.keys(): Use Cases

Or else we can also utilize to get the number of properties stored in the object.

```
1  const count = Object.keys(person).length;
2  console.log(count); // Output: 3
```

Since Object.keys() return an array containing all the keys, we can use array property *length* to get the number of properties stored.

Data Retrieval: Object.values()

Just like Object.keys() return all the keys of the object, Object.values() retrieves all the values of an object's enumerable properties as an array.
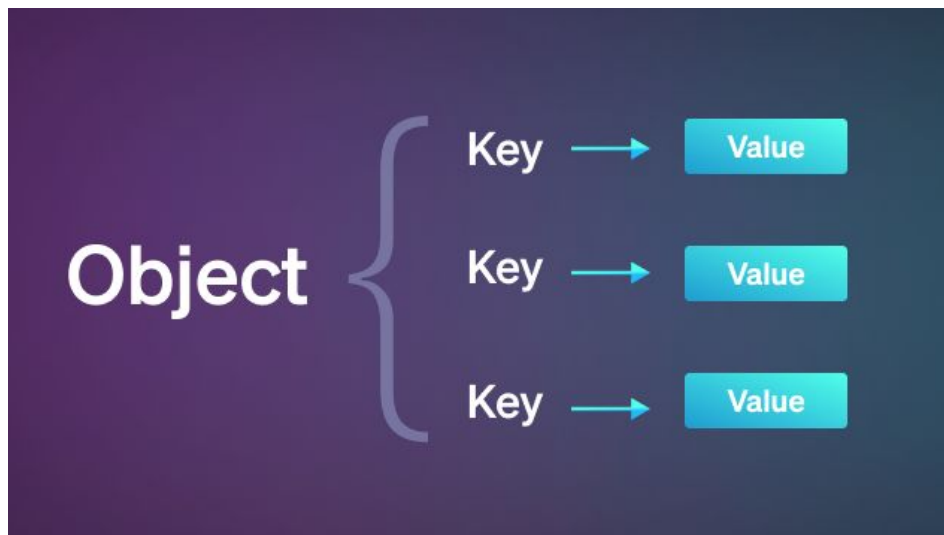
We can retrieve values of properties in an object using its keys. *What if we can access these values directly.*

Here comes *Object.values()* handy.

Data Retrieval: Object.values()

Let's understand it with an example:-

```
1   const values = Object.values(person);
2
3   console.log(values);
4   // Output: ['Alice', 25, 'New York']
```

Here we retrieved all the values of properties stored in the Object.

Data Retrieval: Use Cases

We can iterate all values using forEach loop.

```
1  Object.values(person).forEach(value => {
2      console.log(value);
3  });
4  // Output:
5  // Alice
6  // 25
7  // New York
```

Here we retrieved all the values of properties stored in the Object.

Data Retrieval: Object.entries()

What if we need both key and values?? We can retrieve both key and values using Object.entries.
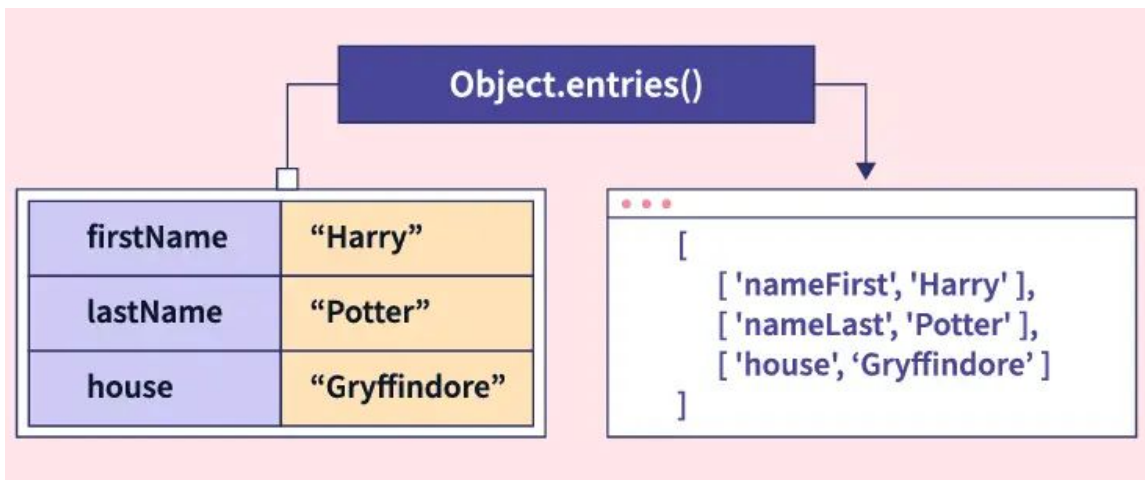
Object.entries() converts an object into an array of key-value pairs for easy looping and modification.

Data Retrieval: Object.entries()

Let's understand it with an example:-

```
1   const character = {
2       firstName: "Harry",
3       lastName: "Potter",
4       house: "Gryffindor"
5   };
6
7   // Convert object into an array of key–value pairs
8   const entriesArray = Object.entries(character);
9
10  console.log(entriesArray);
11
12  // Output:
13  // [
14  //    ['firstName', 'Harry'],
15  //    ['lastName', 'Potter'],
16  //    ['house', 'Gryffindor']
17  // ]
```

Object enumerable properties

Converted into array

Object.entries(): Use Cases

We use Object.entries() where we need both key and value pairs at the same time.

```javascript
const user = {
    firstName: "Harry",
    lastName: "Potter",
    house: "Gryffindor"
};

for (const [key, value] of Object.entries(user)) {
  console.log(`${key}: ${value}`);
}
// Output:
// firstName: Harry
// lastName: Potter
// house: Gryffindor
```

Oh! Great we can get both key, value pairs at the same time.

Data Retrieval: Use Cases

We can iterate all values using forEach loop.

```
Object.values(person).forEach(value => {
    console.log(value);
});
// Output:
// Alice
// 25
// New York
```

Here we retrieved all the values of properties stored in the Object.

Data Creation: Object.fromEntries()

We learned that we can convert an Object to an Array using Object.fromEntries(),

what if we need other way around?

Object.fromEntries() `JS`

[['firstName', 'Harry'], ['lastName', 'Potter'], ['house', 'Gryffindor']]

```
Object.fromEntries(array)
```

{firstName: "Harry", lastName: "Potter", house: "Gryffindor"}

We use *Object.fromEntries() to convert an Array to an Object*.

Object.fromEntries(): Example

Let's Understand it with an example:-

```javascript
const entries = [
    ["firstName", "Harry"],
    ["lastName", "Potter"],
    ["house", "Gryffindor"]
];

const character = Object.fromEntries(entries);

console.log(character);
// Output:
// {
//   firstName: 'Harry',
//   lastName: 'Potter', house: 'Gryffindor'
// }
```

Now you should be able to convert array to object and object to array with ease.

Shallow Copy vs Deep Copy

Not all Copies are equal

Copying an object in JavaScript isn't always simple—some copies stay connected to the original, while others create a completely separate version.

Contents in memory

Memory address: 28208

We can create copy in two ways

Shallow Copy

Deep Copy
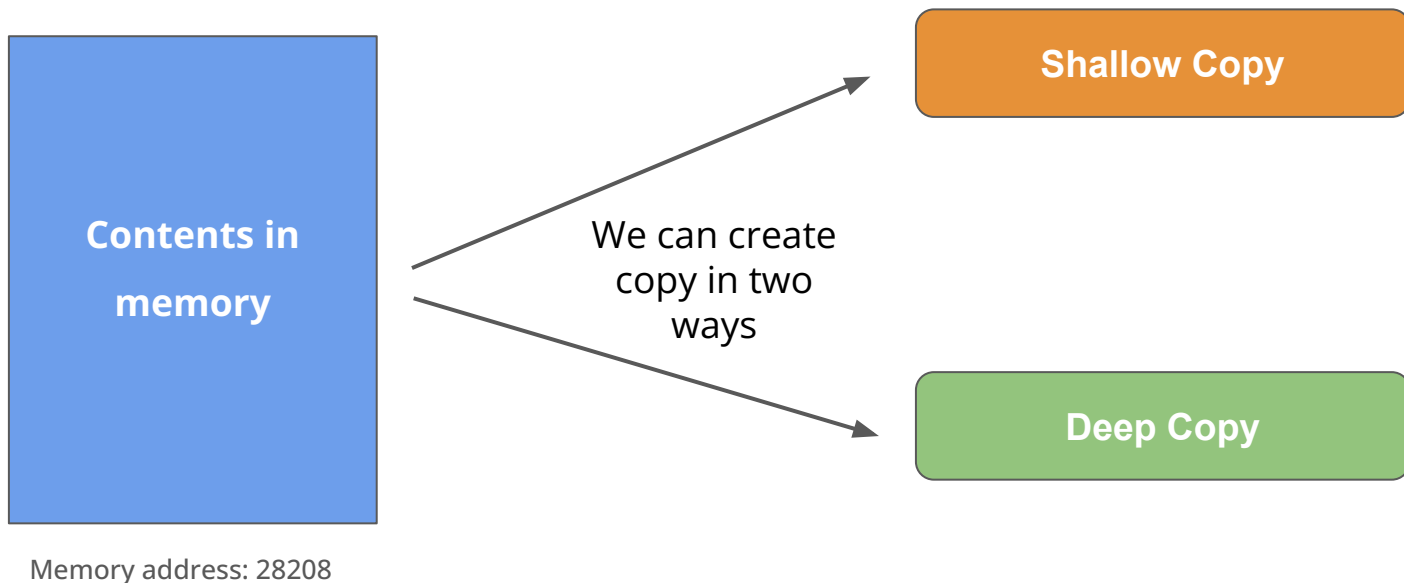
Reference Variables points to memory

In JavaScript, objects are stored in memory, and each object is assigned a memory location. The memory address of that location is stored in a reference variable.

Reference variable 1	holds →	28208		Contents in memory

Location where contents are stored

Memory address: 28208
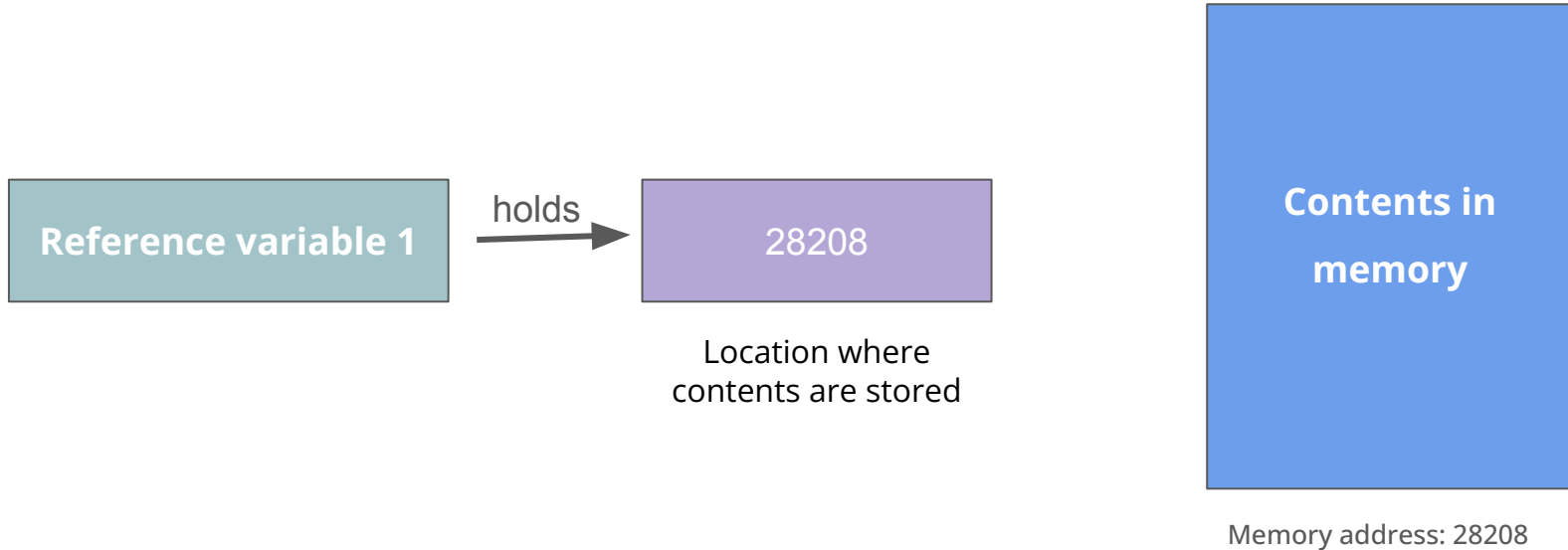
Storing address references

Thus reference variable points to the memory location, it doesn't store the content itself.
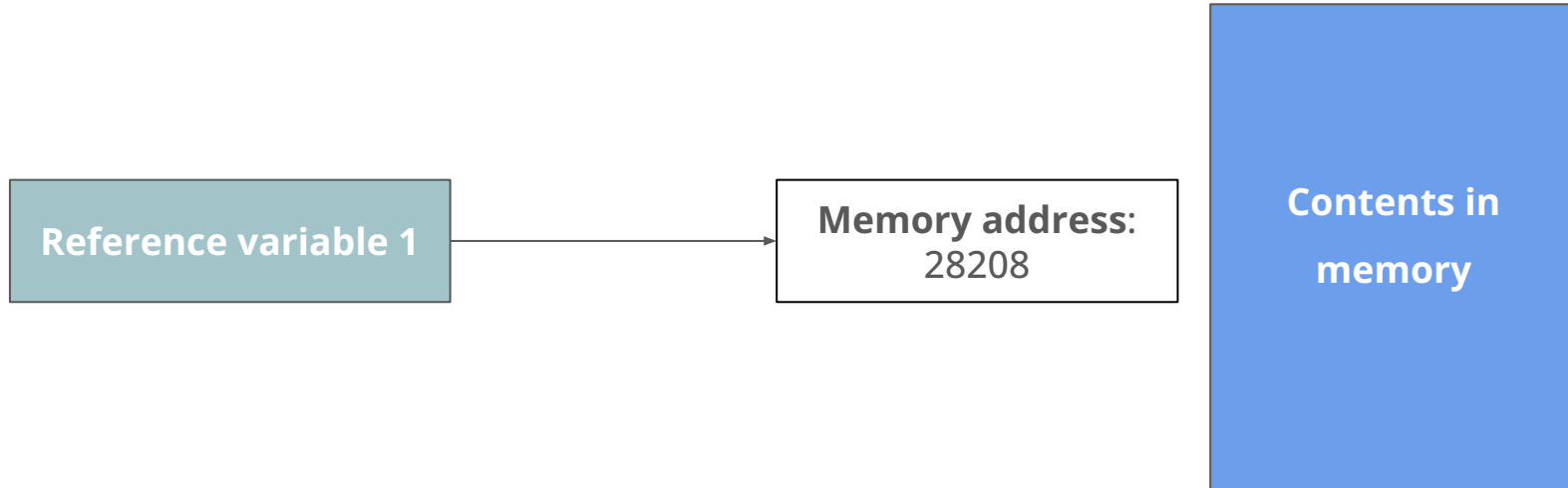
Copying reference values

Copying reference values doesn't mean copying the content itself but memory reference to that content. Such a copy is called **Shallow Copy**.
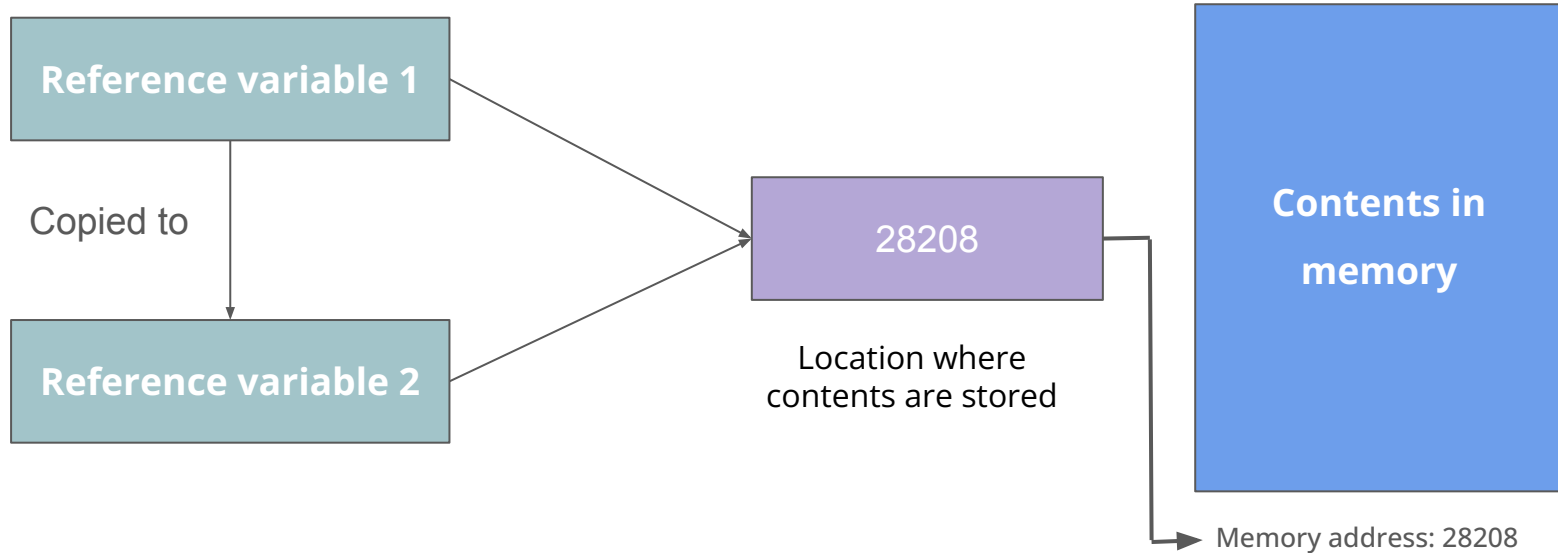
What if we updated the content?

Even if we update our content, still both the reference variables would point to same memory location and in turn the updated content.

Reference variable 1

Copied to

Reference variable 2

28208

Location where contents are stored

Updated contents in memory

Memory address: 28208
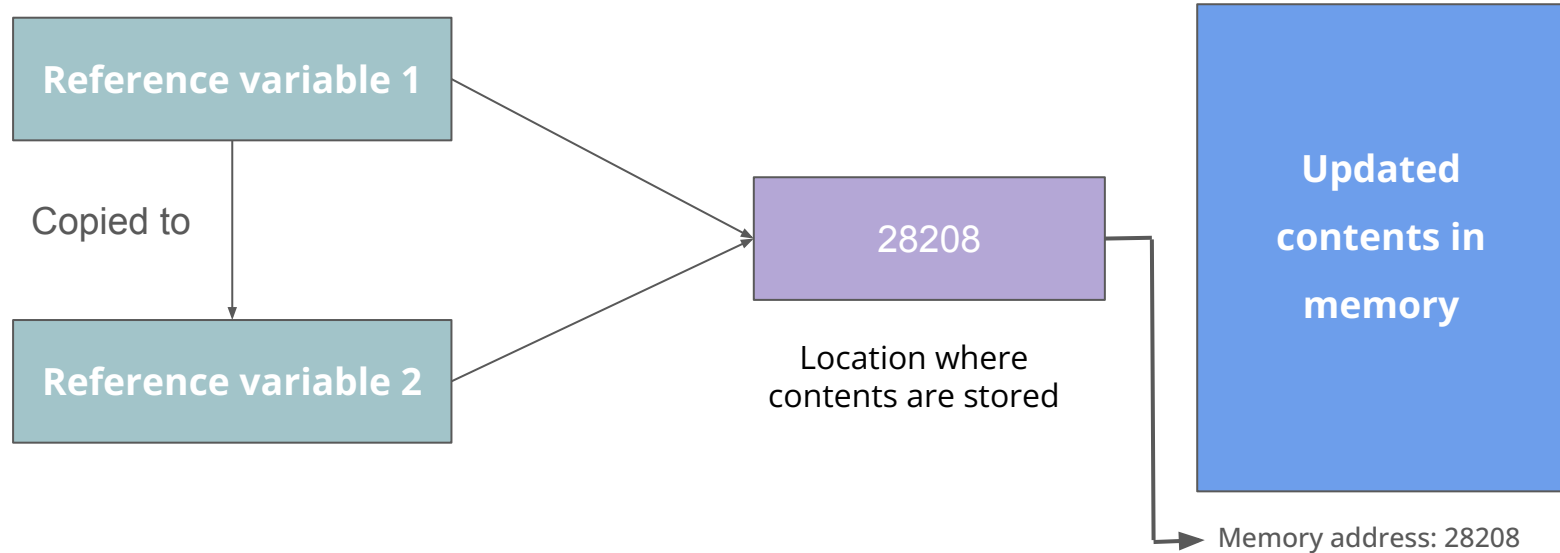
Shallow Copy: Copying references

A shallow copy means copying the memory reference, not the actual content.

```
1    // Original object
2    let person = { name: "John", age: 30 };
3
4    // Shallow copy: copying the reference to the original object
5    let anotherPerson = person;
6
7    // Modifying the copied object also modifies the original
8    anotherPerson.age = 31;
9
10   console.log(person.age);
11   // Output: 31 (original object is affected)
12
13   console.log(anotherPerson.age);
14   // Output: 31 (both point to the same memory)
```

Same content

Memory: 2432

person

anotherPerson

Both variables point to the same memory. Modifying one affects the other.

Copying the Content itself

What if we made a copy of content in the memory and then stored that new memory location in another reference variable. It is called **Deep Copy**.

Reference variable 1	

Memory address: 18202

Memory address: 28208

Contents in memory	copied	Contents in memory

Reference variable 2	
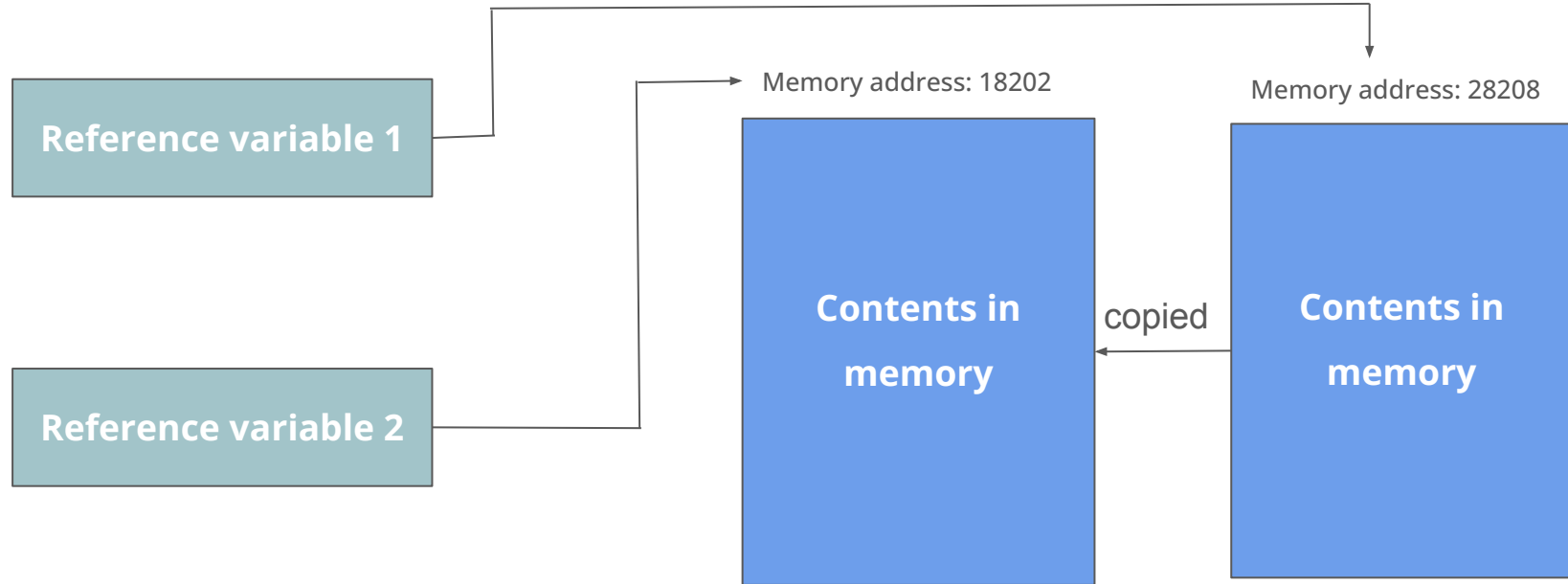
Deep Copy: Copying Content

One way of making a deep copy is by using JSON methods i.e. JSON.stringify() and JSON.parse()

```javascript
1   let person = {
2       name: "John",
3       age: 30,
4       address: { city: "New York", zip: "10001" }
5   };
6
7   // Deep copy using JSON methods
8   let deepCopy = JSON.parse(JSON.stringify(person));
9
10  deepCopy.address.city = "Los Angeles";
11
12  console.log(person.address.city);
13  // Output: "New York" (original is unchanged)
14
15  console.log(deepCopy.address.city);
16  // Output: "Los Angeles" (deep copy is independent)
```
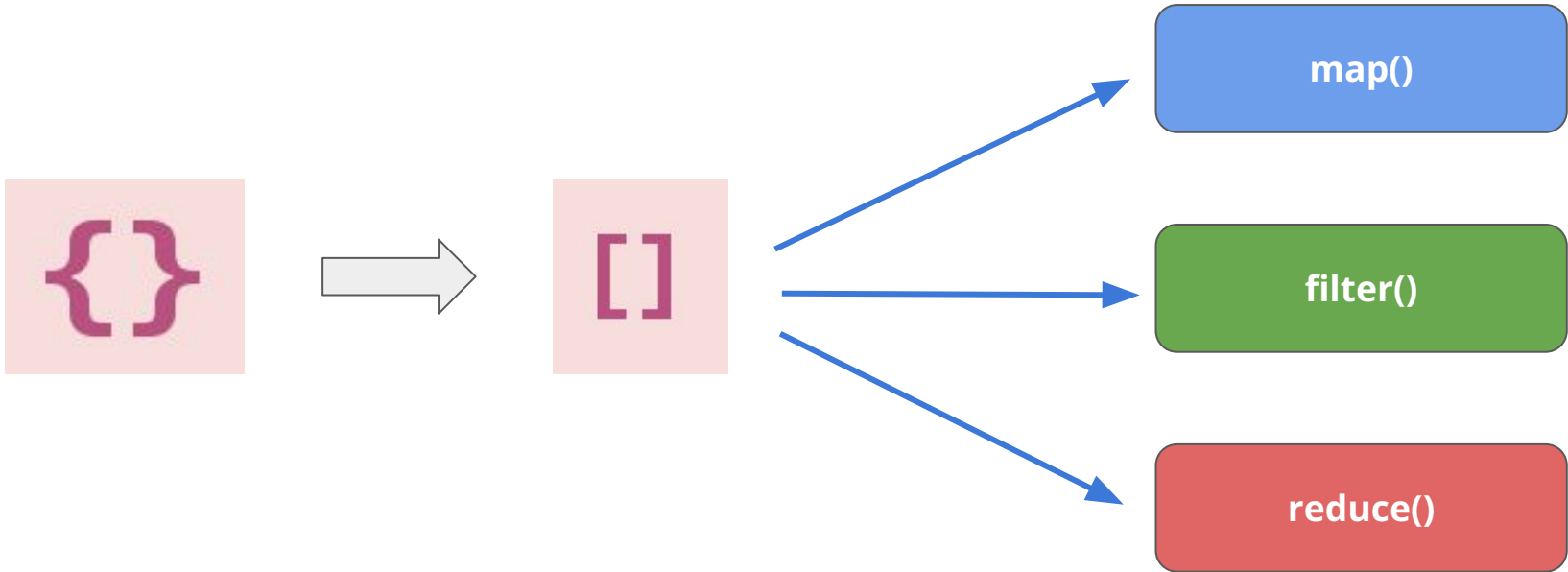
Changing our deep copy, wouldn't affect the original copy

Original copy remains Unchanged

Practical Applications

Unlocking the Power of Array Methods

Object utilities like Object.keys(), Object.values(), etc transform objects into array-like structures, making possible to use array methods such as .map(), .filter(), and .reduce().

Object.keys() with map()

Object.keys() gets the keys of an object, and when used with .map(), it lets you easily transform those keys into a new array.

```javascript
const person = {
    firstName: "Harry",
    lastName: "Potter",
    house: "Gryffindor"
};


// Use Object.keys() with .map() to transform the keys
const keys = Object.keys(person).map((key) => {
    return key;
});

console.log(keys);
// Output: ['firstName', 'lastName', 'house']
```

Since most of the Object utilities return an array, we can iterate those returned values using array methods easily.

Object.values() with map()

Let's iterate values using map:-

```javascript
const person = {
    firstName: "Harry",
    lastName: "Potter",
    house: "Gryffindor"
};

// Use Object.values() with .map()
const values = Object.values(person).map((value) => {
    return value;
});

console.log(values);
// Output: ['Harry', 'Potter', 'Gryffindor']
```

Object.keys() with map()

Object.keys() gets the keys of an object, and when used with .map(), it lets you easily transform those keys into a new array.

```javascript
const person = {
    firstName: "Harry",
    lastName: "Potter",
    house: "Gryffindor"
};


// Use Object.keys() with .map() to transform the keys
const keys = Object.keys(person).map((key) => {
    return key;
});

console.log(keys);
// Output: ['firstName', 'lastName', 'house']
```

Since most of the Object utilities return an array, we can iterate those returned values using array methods easily.

Filtering Object Entries

We can use Object.entries() along with .filter() to select specific key-value pairs from an object based on their keys.

```javascript
const person = {
    firstName: "Harry",
    lastName: "Potter",
    house: "Gryffindor"
};

// Use Object.entries() and filter to get only firstName and lastName
const filteredEntries = Object.entries(person).filter(([key, value]) =>
    key === 'firstName' || key === 'lastName'
);

console.log(filteredEntries);
// Output: [['firstName', 'Harry'], ['lastName', 'Potter']]
```

Iterating over filtered entries using for...in

We can further iterate over filtered entries using for...in loop:-

```javascript
const person = {
    firstName: "Harry",
    lastName: "Potter",
    house: "Gryffindor"
};

// Use Object.entries() and filter to get only firstName and lastName
const filteredEntries = Object.entries(person).filter(([key, value]) =>
    key === 'firstName' || key === 'lastName'
);

// Use for...of loop to iterate over the filtered entries
for (const [key, value] of filteredEntries) {
    console.log(`${key}: ${value}`);
}

// Output:
// firstName: Harry
// lastName: Potter
```

We can convert object into desired array form using Object Utilities and then we can iterate them just like we iterate arrays.

In Class Questions

References

1. **MDN Web Docs - JavaScript**: Comprehensive and beginner-friendly documentation for

   JavaScript.

   https://developer.mozilla.org/en-US/docs/Web/JavaScript

2. **Eloquent JavaScript**: A free online book covering JavaScript fundamentals and advanced topics.

   https://eloquentjavascript.net/

3. **JavaScript.info**: A modern guide with interactive tutorials and examples for JavaScript learners.

   https://javascript.info/

4. **freeCodeCamp JavaScript Tutorials**: Free interactive lessons and coding challenges to learn

   JavaScript.

   https://www.freecodecamp.org/learn/

Thanks for watching!