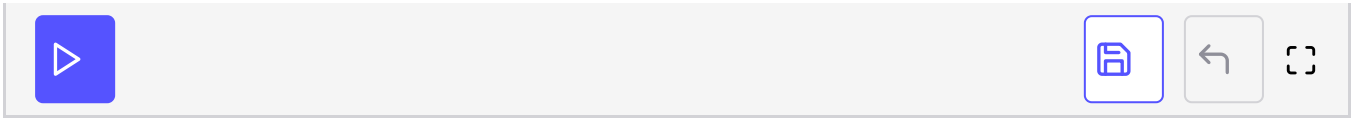
Object.keys

Learn the most powerful and commonly used Object method, 'Object.keys'. We'll discover how it enables us to work with objects as if they were arrays and how we can then use array methods to work with object data.

We've been talking about arrays until now. Let's move to objects. Like arrays, JavaScript objects have several functions available to them. There's one that is by far more useful than any of the others: `Object.keys`.

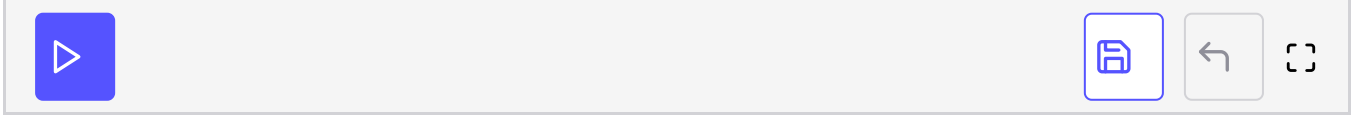
One way to loop through each of an object's properties is to use a `for-in` loop. This will iterate through the properties and allow us to work with the object's keys.

```
const obj = {
    val1: 'abc',
    val2: 'def'
};

for(const key in obj) { // const works here, but not in normal for-loops
    console.log(key, obj[key]);
}
// -> val1 abc
// -> val2 def
```



`Object.keys` allows us to do more. It takes in an object as a parameter and returns an array that contains the keys of the object.

```
const obj = {
    val1: 'abc',
    val2: 'def'
};

const keys = Object.keys(obj);
console.log(keys); // -> ['val1', 'val2]
```



That's really all there is to this function. Why it's so powerful is that we can now use the array methods on this new array that we have. Remember that the array methods `map`, `filter`, and `reduce` return arrays. This means we can chain these function calls.

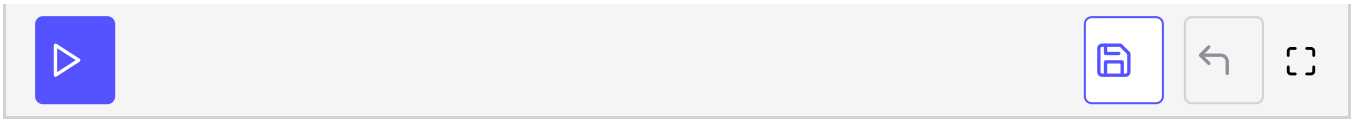
Note that the order of the keys in our new array is not determinate. Different engines will implement `Object.keys` differently and we can't rely on any observed pattern in how the keys come out. Often the keys in the array will be in the same order in which they were inserted, but they may not be. We should expect the keys inserted into our array in any order.

If we do need the keys in a certain order, it's best to use a sort function.

Chaining Array Methods

If we need to work with the values of the object instead of the keys, it's easy. We can get an array of the keys and map it to the values.

```
const obj = {
    val1: 'abc',
    val2: 'def'
};
const values = Object.keys(obj).map(key => obj[key]);
console.log(values); // -> ['abc', 'def']
```

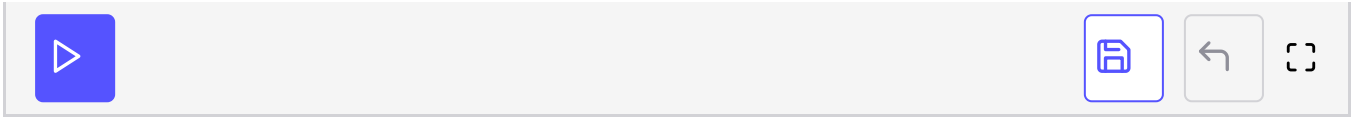


We're using `Object.keys` to get an array of the keys of `obj`. We call `map` on that array and turn each `key` into `obj[key]`, the value of that key in `obj`. The result is the values of `obj`.

We can do more than this. What if we want to get all values in an object that are not `null`?

```
const obj = {
    val1: 'abc',
    val2: 'def',
    val3: '',
    nullProperty: null
};
```

```
const values = Object.keys(obj)
  .map(key => obj[key])
  .filter(value => value !== null);

console.log(values); // -> ['abc', 'def', '']
```



We can chain these method calls indefinitely. It's not uncommon to see or write 3 different method calls on the same array. We can start with an array, filter out values we don't like, transform the remaining values, filter more, transform again, and then reduce them into one final item.

That's it for `Object.keys` and method chaining.

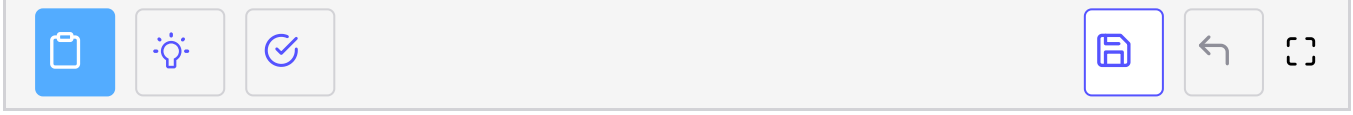
Test Yourself

Feel free to test yourself with these problems.

Problem 1:

Turn this object into a single value. Multiply each key by 3 and add them up. Try doing it in a single method chain!

```
const obj = {
    val1: 4,
    val2: 5,
    val3: 6,
};

// Finish this:
const sum =
```



Problem 2:

Turn this object into a single value. Filter out all items with a negative cost. Multiply the cost of each remaining item times the amount purchased and add it all up. Try doing it in a single method chain!

```
const obj = {
    groceries: {
```

```
        cost: 33.22,
        amount: 1
    },
    rent: {
        cost: 899.00,
        amount: 1,
    },
    paycheck: {
        cost: -2000,
        amount: 2
    },
    restaurantBills: {
        cost: 20,
        amount: 4
    }
};

// Finish this:
const sum =
```