Cheat Sheet – Iterators & Generators

Iterators

Iterators probably sound more complex than they are. Basically an iterator has a function – next() – which allows you to output values step by step.

```
let array = [1, 2, 3];

let iterator = array[Symbol.iterator]();
console.log(iterator.next()); // prints {done: false, value: 1}
```

Calling next prints the current state (done => false or true, depending on the amount of values left) and the current value.
The cool thing about iterators is, that you can implement them in your own objects.

```
let iterableObj = {
    [Symbol.iterator]() {
        let nextVal = 5;

        return {

            next() {

                return {

                    value: nextVal++,

                    done: false

                }

            }

        }

    }

};
let iterator = iterableObj[Symbol.iterator]();

console.log(iterator.next()); // {done: false, value: 5}
```

Note that the iterator specified here would run forever, since done is never set to true.
More information can be found here:
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Iteration_protocols

Generators

Generators are functions which don't necessarily run to the end upon execution.
Instead, upon each call they **yield** a value. A generator is created by adding an asterisk in front of the function name.

```
function *select() {
    yield 'House Data';
    yield 'Person Data';
}
```

When executing a function they don't return a value immediately, instead an iterator is returned. This iterator may then be used to access the returned values step by step.

```
let iterator = select();
console.log(iterator.next()); // prints House Data
console.log(iterator.next()); // prints Person Data
console.log(iterator.next()); // prints undefined
```

Of course you may also pass arguments to generators and use them in the function body.
More information on generators can be found here:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*