Iterators, Generators and Promises

- Iterators
- Generators
- Yielding in Generators
- throw and return
- Promises
- Promise Features

Iterators

```
let ids = [9000, 9001, 9002];
console.log(typeof ids[Symbol.iterator] );
```

Q What shows in the console?

A function

```
let ids = [9000, 9001, 9002];
let it = ids[Symbol.iterator]();
console.log(it.next());
```

Question

What shows in the console?

Answer

{done: false, value: 9000}

```
let ids = [9000, 9001, 9002];
let iter = ids[Symbol.iterator]();
iter.next();
iter.next();
console.log(iter.next());
```

Question

What shows in the console?

Answer

{done: false, value: 9002}

```
let ids = [9000, 9001, 9002];
let iter = ids[Symbol.iterator]();
iter.next();
iter.next();
iter.next();
console.log(iter.next());
```

Question

What shows in the console?

Answer

{done: true, value: undefined}

```
let idMaker = {
    [Symbol.iterator]() {
        let nextId = 8000;
        return {
            next() {
                return {
                    value: nextId++,
                    done: false
                };
            }
        };
    }
};
let it = idMaker[Symbol.iterator]();
console.log(it.next().value);
console.log(it.next().value);
```

Question

What shows in the console?

Answer

8000
8001

```
let idMaker = {
    [Symbol.iterator]() {
        let nextId = 8000;
        return {
            next() {
                return {
                    value: nextId++,
                    done: false
                };
            }
        };
    }
};
for (let v of idMaker) {
    if (v > 8002) break;
    console.log(v);
}
```

Question

What shows in the console?

Answer

8000
8001
8002

```
let idMaker = {
    [Symbol.iterator]() {
        let nextId = 8000;
        return {
            next() {
                let value = nextId>8002?undefined:nextId++;
                let done = !value;
                return { value, done };
}};}};

for (let v of idMaker)
    console.log(v);
```

Q What shows in the console?

A

8000
8001
8002

```
let ids = [8000, 8001, 8002];

function process(id1, id2, id3) {
    console.log(id3);
}

process(...ids);
```

Q What shows in the console?

A 8002

Generators

```
function *process() {
    yield 8000;
    yield 8001;
}
let it = process();
console.log(it.next());
```

Question

What shows in the console?

Answer

{done: false, value: 8000}

```
function *process() {
    yield 8000;
    yield 8001;
}
let it = process();
it.next();
console.log(it.next());
```

Question

What shows in the console?

Answer

{done: false, value: 8001}

```
function *process() {
    yield 8000;
    yield 8001;
}
let it = process();
it.next();
it.next();
console.log(it.next());
```

Question

What shows in the console?

Answer

{done: true, value: undefined}

```
function *process() {
    let nextId = 7000;
    while(true)
        yield(nextId++);
}
let it = process();
it.next();
console.log(it.next().value);
```

Question

What shows in the console?

Answer

7001

```
function *process() {
    let nextId = 7000;
    while(true)
        yield(nextId++);
}

for (let id of process()) {
    if (id > 7002) break;
    console.log(id);
}
```

Question

What shows in the console?

Answer

7000
7001
7002

Yielding in Generators

```
function *process() {
    yield 8000;
}

let it = process();
console.log(it.next());
```

Question

What shows in the console?

Answer

{done: false, value: 8000}

```
function *process() {
    yield;
}

let it = process();
console.log(it.next());
```

Question

What shows in the console?

Answer

{done: false, value: undefined}

```
function *process() {
    let result = yield;
    console.log(`result is ${result}`);
}

let it = process();
it.next();
it.next(200);
```

Question

What shows in the console?

Answer

result is 200

```
function *process() {
    let result = yield;
    console.log(`result is ${result}`);
}

let it = process();
it.next();
console.log(it.next(200));
```

Question

What shows in the console?

Answer

result is 200

{done: true, value: undefined}

```
function *process() {
    let newArray = [yield, yield, yield];
    console.log(newArray[2]);
}

let it = process();
it.next();
it.next(2);
it.next(4);
it.next(42);
```

Question

What shows in the console?

Answer

42

```
function *process() {
    let value = 4 * yield 42;
    console.log(value);
}

let it = process();
it.next();
it.next(10);
```

Question

What shows in the console?

Answer

Syntax Error

```
function *process() {
    let value = 4 * (yield 42);
    console.log(value);
}

let it = process();
it.next();
it.next(10);
```

Question

What shows in the console?

Answer

40

```
function *process() {
    yield 42;
    yield [1,2,3];
}

let it = process();
console.log(it.next().value);
console.log(it.next().value);
console.log(it.next().value);
```

Question

What shows in the console?

Answer

42
[1,2,3]
undefined

```
function *process() {
    yield 42;
    yield* [1,2,3];
}

let it = process();

console.log(it.next().value);
console.log(it.next().value);
console.log(it.next().value);
console.log(it.next().value);
console.log(it.next().value);
```

Question

What shows in the console?

Answer

42
1
2
3
undefined

throw and return

```
function *process() {
    try {
        yield 9000;
        yield 9001;
        yield 9002;
    }
    catch(e) {

    }
}

let it = process();
console.log(it.next().value);
console.log(it.throw('foo'));
console.log(it.next());
```

Question

What shows in the console?

Answer

9000
{done:true, value:undefined}
{done:true, value:undefined}

```
function *process() {
    yield 9000;
    yield 9001;
    yield 9002;
}

let it = process();
console.log(it.next().value);
console.log(it.throw('foo'));
console.log(it.next());
```

Question

What shows in the console?

Answer

9000
Exception: foo
(execution terminates)

```
function *process() {
    yield 9000;
    yield 9001;
    yield 9002;
}

let it = process();
console.log(it.next().value);

// Firefox only (currently)
console.log(it.return('foo'));

console.log(it.next());
```

Question

What shows in the console?

Answer

9000
{value: "foo", done: true}
{value: undefined, done: true}

Promises

```
function doAsync() {
    let p = new Promise(function (resolve, reject) {
        console.log('in promise code');
        setTimeout(function () {
            console.log('resolving...');
            resolve();
        }, 2000);
    });
    return p;
}

let promise = doAsync();
```

Q What shows in the console?

A
in promise code
(2 second delay)
resolving...

```
function doAsync() {
    let p = new Promise(function (resolve, reject) {
        console.log('in promise code');
        setTimeout(function () {
            console.log('rejecting...');
            reject();
        }, 2000);
    });
    return p;
}

let promise = doAsync();
```

Q What shows in the console?

A
in promise code
(2 second delay)
rejecting...

```
function doAsync() {
    // returns a Promise, will be rejected
}

doAsync().then(function () {
    console.log('Fulfilled!');
},
function () {
    console.log('Rejected!');
});
```

Q What shows in the console?

A in promise code
(wait for resolution)
Rejected!

```
function doAsync() {
    // returns a Promise, will be resolved
}

doAsync().then(function () {
        console.log('Fulfilled!');
    },
    function () {
        console.log('Rejected!');
    }
);
```

Q What shows in the console?

A in promise code
(wait for resolution)
Fulfilled!

```
function doAsync() {
    // returns a Promise, will be rejected using:
    // reject('Database Error');
}

doAsync().then(function (value) {
    console.log('Fulfilled! ' + value);
},
function (reason) {
    console.log('Rejected! ' + reason);
});
```

Q What shows in the console?

A in promise code
(wait for resolution)
Rejected! Database Error

```
function doAsync() {
    // returns a Promise, will be resolved using:
    // resolve('OK');
}

doAsync().then(function (value) {
    console.log('Fulfilled! ' + value);
},
function (reason) {
    console.log('Rejected! ' + reason);
});
```

Q What shows in the console?

A in promise code
(wait for resolution)
Fulfilled! OK

```
function doAsync() {
    // returns a Promise, will be resolved using:
    // resolve('OK');
}

doAsync().then(function (value) {
    console.log('Fulfilled! ' + value);
    return 'For Sure';
}).then(function(value) {
    console.log('Really! ' + value);
});
```

Q What shows in the console?

A

in promise code
(wait for resolution)
Fulfilled! OK
Really! For Sure

```
function doAsync() {
    // returns a Promise, will be rejected using:
    // reject('No Go');
}

doAsync().catch(function (reason) {
    console.log('Error: ' + reason);
});
```

Q What shows in the console?

A (wait for resolution)
Error: No Go

More Promise Features

```
function doAsync() {
    let p = new Promise(function (resolve, reject) {
        console.log('in promise code');
        setTimeout(function () {
            resolve( getAnotherPromise() );
        }, 2000);
    });
    return p;
}

doAsync().then(function () { console.log('Ok') },
               function () { console.log('Nope')});
```

Q What shows in the console?

A in promise code
Nope

```
function doAsync() {
    return Promise.resolve('Some String');
}

doAsync().then(
    function (value) { console.log('Ok: ' + value) },
    function (reason) { console.log('Nope: ' + reason)}
);
```

Q What shows in the console?

A Ok: Some String

```
function doAsync() {
    return Promise.reject('Some Error');
}

doAsync().then(
    function (value) { console.log('Ok: ' + value) },
    function (reason) { console.log('Nope: ' + reason)}
);
```

Q What shows in the console?

A Nope: Some Error

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.all([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);

// assume p1 resolves after 3 seconds,
// assume p2 resolves after 5 seconds
```

Q What shows in the console?

A (5 second delay)
Ok

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.all([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);

// assume p1 resolves after 1 second,
// assume p2 is rejected after 2 seconds
```

Q What shows in the console?

A (2 second delay)
Nope

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.all([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);

// assume p1 is rejected after 3 second,
// assume p2 is rejected after 5 seconds
```

Q What shows in the console?

A (3 second delay)
Nope

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.race([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);

// assume p1 resolves after 3 second,
// assume p2 resolves after 5 seconds
```

Q What shows in the console?

A (3 second delay)
Ok

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.race([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);

// assume p1 is rejected after 3 second,
// assume p2 resolves after 5 seconds
```

Q What shows in the console?

A (3 second delay)
Nope

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.race([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);

// assume p1 resolves after 4 second,
// assume p2 is rejected after 5 seconds
```

Q What shows in the console?

A (4 second delay)
Ok

Summary

Iterators

```
let ids = [9000, 9001, 9002];
let iter = ids[Symbol.iterator]();
iter.next();
iter.next();
console.log(iter.next());
```

Summary

Generators

```
function *process() {
    let nextId = 7000;
    while(true)
        yield(nextId++);
}
let it = process();
it.next();
console.log(it.next().value);
```

Summary

Yielding

```
function *process() {
  let newArray = [yield, yield, yield];
  console.log(newArray[2]);
}
```

Summary

throw and return

```
function *process() {
    try {
        yield 9000;
        yield 9001;
        yield 9002;
    }
    catch(e) {

    }
}

let it = process();
console.log(it.next().value);
console.log(it.throw('foo'));
console.log(it.next());
```

Summary

Promises

```
function doAsync() {
    let p = new Promise(function (resolve, reject) {
        console.log('in promise code');
        setTimeout(function () {
            console.log('resolving...');
            resolve();
        }, 2000);
    });
    return p;
}

let promise = doAsync();
```

Summary

Promise.all() and Promise.race()

```
let p1 = new Promise(...);
let p2 = new Promise(...);

Promise.all([p1, p2]).then(
    function (value) { console.log('Ok') },
    function (reason) { console.log('Nope') }
);
```