UNDERSTANDING ASYNC AWAIT

"Async/Await keywords allows us to **pause the execution of functions** and this in turn writes asynchronous code that reads like synchronous code."

- MPJ, FunFunFunction

GENERATORS!

GENERATOR PRIMER

```
function* generator(i) {
  yield i;
  yield i + 10;
}

var gen = generator(10);
console.log(gen.next().value);
// expected output: 10
console.log(gen.next().value);
// expected output: 20
// https://developer.mozilla.org/en-US/docs/Web/JavaScript/Refere
```

INFINITE GENERATOR

```
function* generator(i) {
    count = 0
    while(true) {
        count = i + count
        yield count; // pauses here
    }
}
var gen = generator(10);
console.log(gen.next().value);
// expected output: 10

console.log(gen.next().value);
// expected output: 20
console.log(gen.next().value);
// expected output: 30
```

REAL WORLD EXAMPLE

```
function foo(x,y,cb) {
    ajax(
        "http://some.url.1/?x=" + x + "&y=" + y,
        cb
    );
}

foo( 11, 31, function(err,text) {
    if (err) {
        console.error( err );
    }
    else {
        console.log( text );
    }
} );
```

REAL WORLD EXAMPLE W GENERATORS

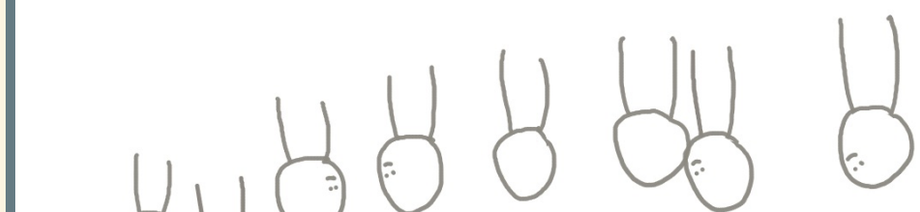
```
function foo(x,y) {
    // Request is a Promise
    return request(
        "http://some.url.1/?x=" + x + "&y=" + y
    );
}

function *main() {
    try {
        var text = yield foo( 11, 31 );
        console.log( text );
    }
    catch (err) {
        console.error( err );
    }
}
```

ASYNC AWAIT USES PROMISES AND ITERATORS UNDER THE HOOD

WHAT ARE PROMISES?

https://kosamari.com/notes/the-promise-of-a-burger-party



PROMISE-AWARE GENERATOR RUNNER

```
function async(it, context = undefined) {
    // Create iterator if necessary
    let iterator = typeof it === 'function' ? it() : it

    // Pass last result, get next Promise
    let { value: promise } = iterator.next(context)

    if ( typeof promise !== 'undefined' ) {
        promise.then(
            resolved => async(iterator, resolved))
                .catch(error => iterator.throw(error))
                // ^Defer to generator error handling
    }
}
```

USING EXAMPLE

```
/* Usage */
async(function* () { // Generators can't be declared with arrow s
    try {
        // Execution is paused until the yielded promise resolves
        console.log(yield Promise.resolve('A Mortynight Run'))
        // Promises already provide support for concurrent async
        // Execution will not continue until both a & b are fulfi
        let [a,b] = yield Promise.all([
            Promise.resolve('Get Schwifty'),
            Promise.resolve('The Ricks Must Be Crazy')
        ])
        console.log(a + ' ' + b)
        // Rejected promises will be handled by try/catch as if c
        let seasonTwoFinale = yield Promise.reject(new Error('Tam
        // Never executed
        let seasonThree = 'Coming Soon'
```

ChrisChares: gist source

Note Differences

- Using Generators + Async you can use interchangable
- generator are not about pausing stuff, not about it
  - yield is waiting, for the execution
- Generators are in ES6 and have been in node since v4+
- Async/Await is sugar on Generators that return Promises

Sources

- https://medium.freecodecamp.org/demystifying-es6-i
4bdd0b084082
- https://gist.github.com/ChrisChares/1ed079b9a6c98
- https://github.com/getify/You-Dont-Know-JS/blob/master/async%20%26%20performance/ch4.m
generator-runner