Welcome To Advanced NodeJS

Monday, October 12, 2020 10:38 AM

Advanced Node Js

Assignment01

Monday, October 12, 2020 10:41 AM

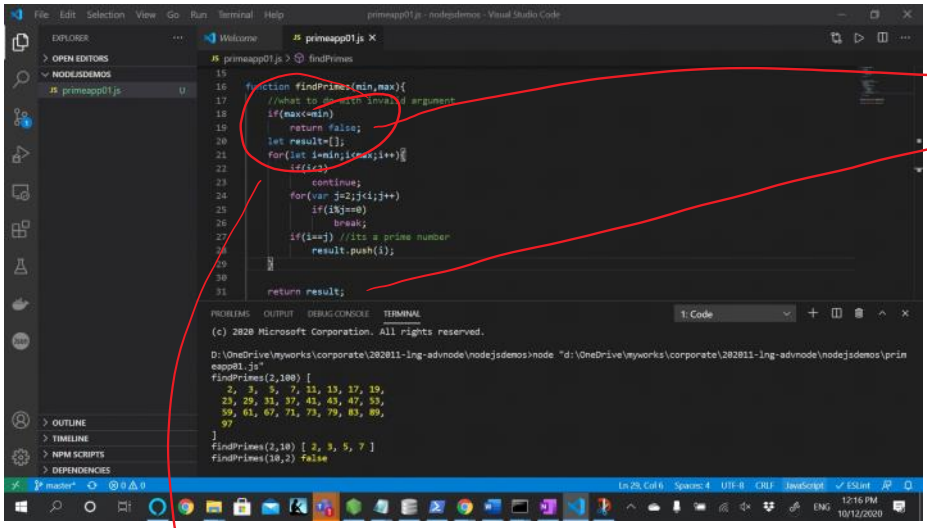
- Create a function to find and return all primes in a given min and max range
  - Example find primes between 2 and 200

- Psudo code of isPrime

```
bool isPrime(int x){

        If(x<2)
                return false;

        for(int i=2;i<x;i++)
                If(x%i==0)
                        return false;

        return true;

}
```

The common problems

Monday, October 12, 2020 12:15 PM



Returning completely different type of values

- Client is forced to check the types

Recommendation!

- **If you function returns an array, always return an array, may be an empty array when you have not value to return instead of returning false or null.**

Don't return a value to indicate an error. If possible **throw exception or any standard Mechanism to indicate error.**

Loose types?

- Javascript as loose (dynamic) types.
- But to create a consistent API we must adhere to some common denomniators

- Example a method may return

```
{
    status: 'success',
    data:[1,2,3,4]
}
```

Or

```
{
    Status:'failed',
    reason:'invalid range'
}
```

Different data

Common denom

Nodejs is Single threaded Asynchronous Programming model

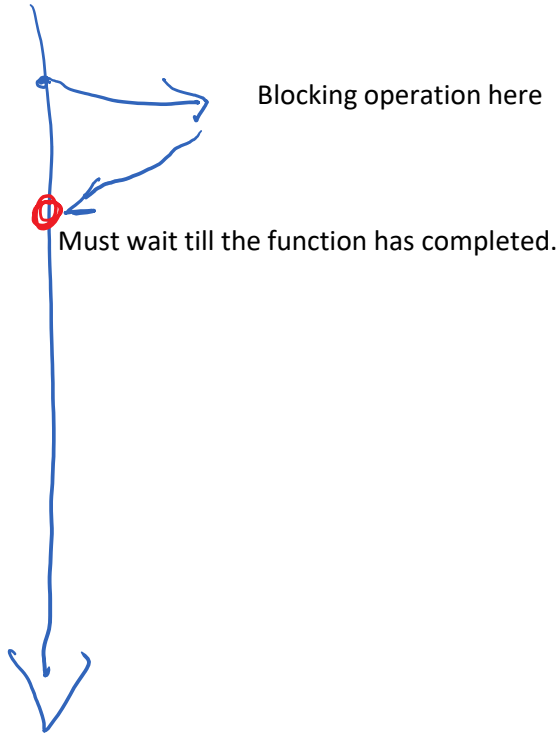
Monday, October 12, 2020 12:30 PM

NodeJS expects your functions to be async by default

- If you function is synchronous for whatever reason, it must be suffixed with the word sync

Note

- **Languages like java and C# using async suffix to mark an asynchronous function.**
- **By default functions are synchronous**
- **NodeJs expects functions to by async by default.**



Javascript Asynchrnous Programming

Monday, October 12, 2020 3:16 PM

- A general paradigm of programming, where we don't need to wait for a function to finish
  - Function returns immediately
  - Continues to work in backgournd
  - Updates the client once it finishes with the help of some kind of call back

Different Types of Asynchrnous Programming Model

1. NodeJS Callback pattern
   a. Callback is not a new concept
   b. NodeJS has a special callback syntax for function : **function callback(err,result);**
      i. **We can use this model anywhere as this is just a pattern and now a NODE JS feature**
      ii. **Most of the NodeJS API follow the same syntax.**

2. **ES2015 Promises**

Assignment 02

Monday, October 12, 2020 12:52 PM

1. Continue with Assignment01 and make the API asynchronous
2. Use Modular approach by separating business and presentation tier

NodeJS Callback Pattern

Monday, October 12, 2020 1:03 PM

1. NodeJS callback architecture

- Nodejs expects your functions not to return using return keyword
- You pass a callback as the last parameter to your function
- Once function finishes it calls the call back
- The callback should take two parameter in order
  - Err
    - Should specificy in case of error
    - Second parameter should be null/undefined
  - Result
    - Err should be null
    - Result should contain the result

```
function findPrimesSync(min,max){

    let result=[];

    return result;
}
```

Should change to

```
function findPrimes(min,max, cb){

    let result=[];
    if(success)
        cb(null, result); //success
    else
        cb('invalid input'); //error
}
```

```
function findPrimes(min, max, cb) {

    setTimeout(() => {
        if (min >= max)
            cb(new Error(`Invalid Range(${min}-${max})`));  //result is undefined
        else {
            let primes = [];
            for (let i = min; i < max; i++)
                isPrime(i, (err, result) => {
                    if (result)
                        primes.push(i);
                });

            cb(null, primes); //first parameter null indicates success
        }

    }, 2); //just to simulate that job may take long time.

}
```

- Is running synchronously as one big chunk of code.
- Once you start, you end only after searching everything
- Not giving any other job time to work
- This is called **selfish** programming

Simulates a long running process

Cooperative Worker Pattern

- A code should allow other codes to work by taking a break
- This should allow vital UI updates and other short worker to complete

How to implement co-operative worker in our code

- Say we are finding all primes between 2 and 500000
- We may take a short break of say 10ms after every 1000 iteration.

Assignment 03

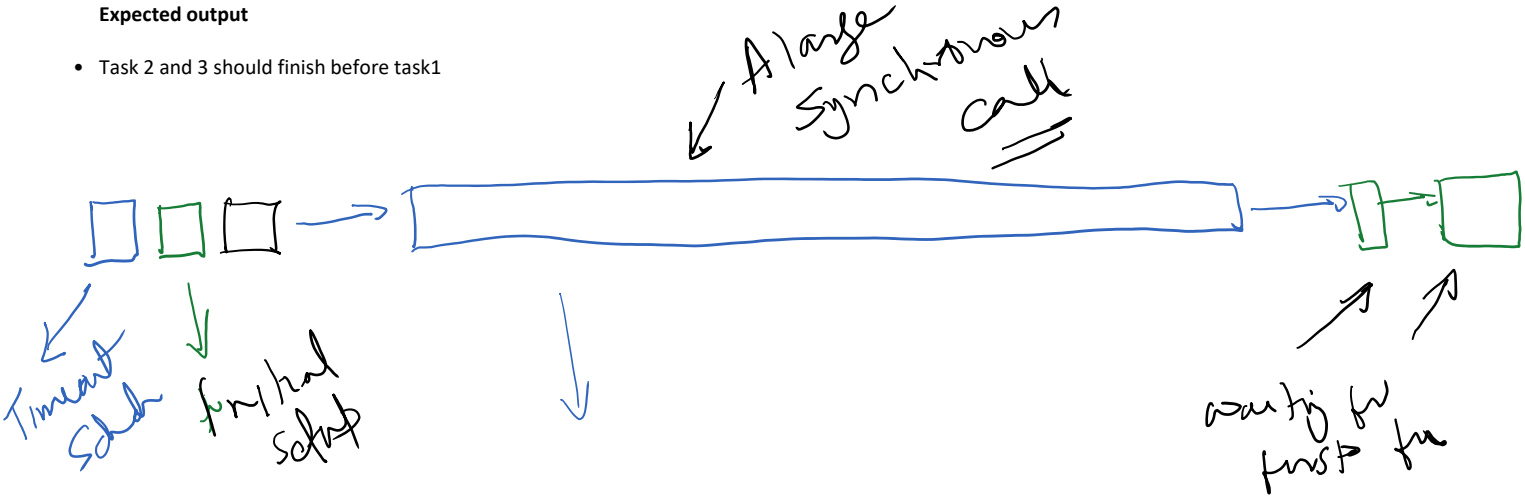
Monday, October 12, 2020 12:52 PM

1. Implement co-operative worker pattern in the findPrimes function shared with you.
    - Take short break of say 2ms after every 1000 number iteration.

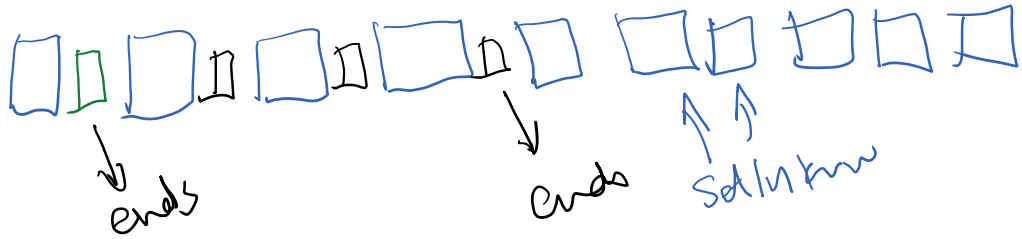
2. The client shouldn't change

Expected output

- Task 2 and 3 should finish before task1



New Design



ES2015 Promises

Monday, October 12, 2020 3:19 PM

- It is not a NodeJS feature but available in general in all javascript programming
- **Evolved much later**
- **NodeJS was already using its own model of programming**

- Many Nodejs libraries are now slowly moving to Promise rather than node callbacks

A Promise

- It's a bulit in ES2015 (Javascript feature)
- **Promise is an object that promises to get some result in future**
  - **Promise also take a callback with two parameters**
    - **These two parameters are again call backs**
      1. **To call when success**
      2. **To call when failed**

- Promise to get a result asynchronously by calling another function

Promise says let me run this code and I will let you know when we are ready

```
let promise= new Promise(   function_that_will_give_you_a_result );
```

```
function  function_that_will_give_you_a_result(   fnResolve, fnReject ){

    …
    if(success)
        fnResolve( result);   //call when you completed successfully
    else
        fnReject(err_details); //call this function when you fail

}
```

This is your business logic

Creating an api — callback vs Promise

```
function findPrimes(   min, max, cb ){
    //business logic
    …
    if(success)
        cb( null, result);
    else
        cb(err_details);

    //This function returns nothing
}
```

No callback passed.

```
function findPrimes(   min, max ){

    let promise=new Promise( (resolve,reject) =>{
        //business logic here
        if(success)
            resolve(  result);
        else
            reject(err_details);

    }
    return promise;
}
```

We handle promise once returned

Consuming The Asynchronous operations

```
//callback example
findPrimes( 2, 100 ,   (err,primes) =>{
```

```
//promise based design

//function doesn't return result. It returns a future promise
```

```
//callback example
findPriMes( 2, 100 ,   (err,primes) =>{

        if(err){
                console.log('err',err); //on failure
        } else{
                Console.log('primes', primes.length);  //on success
        }

});

//we are free to do whatever we want
//the callback will be called sometimes in future
//same callback will get both err and result
```

```
//promise based design

//function doesn't return result. It returns a future promise
let promise= findPrimes(2,100);

//we can set for future when it completes
//if promise is resolved successfully
promise. then( primes=> console.log('primes', promes.length);

//if promise is rejected because of error
promise.catch( err => console.log( 'err', err);


//we can do whatever we want to do. then() and catch() will
execute asynchrnously when promise is resolved/rejected in
future.

//this code will execute immediately.
```

Promises can
Be chained

```
findPrimes(2,100)
        .then(primes=> console.log(primes))
        .catch(err=>console.log(err);
```

Async - Await Keywords

- Since Promise is a javascript feature, javascript has defined a set of keywords that makes working with Promise easy and straight forward.

Assignment 04

Monday, October 12, 2020 3:41 PM

- Convert findPrimes from callback to Promise model
- Write the test application

Assignment05

Monday, October 12, 2020 4:32 PM

Create a long running factorial function.

- Psudo code for factorial

```
int factorial(int n){
        if(n<0) //error

        let fn=1;

        while(n>1)
                fn*=n--;

        return fn;

}
```

Assume factorial is a long running task and needs n*100 ms to complete

1. Create an asynchrnous factorial function that returns in n*100 ms.
   a. It should return a promise

2. Use the factorial function to calculate comination(n,r); psudocode for combination is

```
int combination(int n, int r){
        int fn=factorial(n);

        int fn_r=factorial(n-r);

        int fr=factorial(r);

        return fn/fn_r/fr;
}
```

Comination will not have any delays programmed.
It will be delayed because of factorial