

Node.js Section 4

Merhaba bugün node.js serimize devam ediyoruz ve 4.bölümdeyiz. Bu hafta arrow function nedir bunu anlayarak devam etmemiz gerekecek.

Arrow Function Nedir ?

Bir tane auxiliary topics adında klasör oluşturalım ve içinde arrowFunction.js JavaScript dosyası oluşturalım ve deneyerek oradan anlamaya çalışalım.

AuxiliaryTopics - arrowFunction.js

```
// * Way 1
function greetings(name) {
    console.log("Hello " + name);
}

// * Way 2
let greetings2 = function (name) {
    console.log("Hello " + name);
}

greetings("Burak");
greetings2("Batuhan ")
```

Normalde fonksiyonları bu şekilde uzun uzun yazmamız gerekiyor ama Arrow function ile daha kısa yazabilmekteyiz.

AuxiliaryTopics - arrowFunction.js

```
const arrowFunctionGreeting =(name) => { console.log("Hello " + name) }
arrowFunctionGreeting("Gül");
```

Bu da kısa bir halidir foksiyonumuzun.

Şimdi arrow function nedir neden kullanmalıyızın en basit sebebidir bu. İlerleyen kısımlarda başka sebeplerini de deagineceğiz.

ARROW FUNCTION SYNTAXI

Bir fonksiyonu arrow function syntax i ile tanımladığımızda/deklare ettiğimizde bu deklarasyonu bir variable/değişkene atamamız gereklidir yani fonksiyon bir isme sahip olmalı.

Genel olarak arrow function syntaxı aşağıdaki gibidir

AuxiliaryTopics - arrowFunction.js

```
const myFunction = (param1, param2, ...) => {  
    // function body  
}
```

- myFunction fonksiyonu tutan bir variable/değişkendir.
- myFunction() ile fonksiyonu daha sonra çağrılabılır.
- (param1,param2, ...) fonksiyon parametreleridir
- => fonksiyonun parametrelerinin bittiğini belirtken { ise fonksiyonun gövdesinin başladığını belirtmektedir.(Fonksiyonun gövdesi tek satırdan oluşuyorsa {} yazmasak da olur)

NORMAL FONKSİYONLARI ARROW FUNCTIONA KOLAYCA ÇEVİRME

3 adımda arrow functiona kolayca çevirebiliriz.

- 1- Function keywordünü const keywordü ile değiştirelim.
- 2- = simbolünü fonksiyon isminden önce parametre belirten parantez “()”lerden önce koyalım.
- 3- => simbollerini de parametre belirten parantezlerden “()” sonra koyalım.

```
AuxiliaryTopics - arrowFunction.js
```

```
function greetings(name) {
    return "Hello " + name;
}

// step 1: function keywordünü const ile değiştir
const greetings(name) {
    return "Hello " + name ;
}

// step 2: fonksiyon isminden sonra = koy
const greetings = (name) {
    return "Hello " + name ;
}

// step 3: parametre parantezlerinde "()" sonra => koy
const greetings = (name) => {
    return "Hello " + name ;
}
```

Single Line fonksiyona sahip olduğumuzda aşağıdaki özellikler optional/isteğe bağlıdır.

Single Line fonksiyon ise gövdesi tek satırdan oluşan fonksiyonlardır diyebiliriz.

- Eğer fonksiyonumuzun gövdesi tek satırdan oluşuyorsa fonksiyon gövdesini belirten parantezlerle beraber return keywordünü kelimesini kaldırmalıyız.

```
AuxiliaryTopics - arrowFunction.js
```

```
const greetings = (name) => {
    return "Hello " + name ;
}

const greetings2 = (name2) => "Hello " + name2
console.log(greetings2("Burak")) // Hello Burak
```

- 1 adet parametremiz varsa parametre parantezlerini kaldırabiliriz.

```
AuxiliaryTopics - arrowFunction.js

const greetings = (name) => {
  return "Hello " + name ;
}

const greetings2 = (name2) => "Hello " + name2
console.log(greetings2("Burak")) // Hello Burak

const greetings3 = name3 => "Hello " + name3
console.log(greetings3("Batuhan")) // Hello Batuhan
```

NOT

- Eğer parametremiz yok ise () koymak zorundayız.

```
AuxiliaryTopics - arrowFunction.js

const greetings4 = () => console.log("Hello World")
greetings4() // Hello World
```

- Single Line olmayan fonksiyonlarda fonksiyon gövdesi parantezlerini "{}" koymak zorundayız

```
AuxiliaryTopics - arrowFunction.js

const greetings5 = () => {
  console.log("Hello World!");
  console.log("How are you?");
};
greetings5() // Hello World! How are you?
```

Arrow Functionu seçme kullanma sebeplerimizden birisi parametre olarak fonksiyon alan fonksiyonları kullanırken işimize yaramaları.

Mesela forEach methodu , bir arrayin her bir elemanı üzerinde gezen ve o eleman üzerinde uygulanacak bir fonksiyon alan fonksiyondur.

Örnek olarak aldığı sayının 2 katından 1 çıkaran fonksiyonu

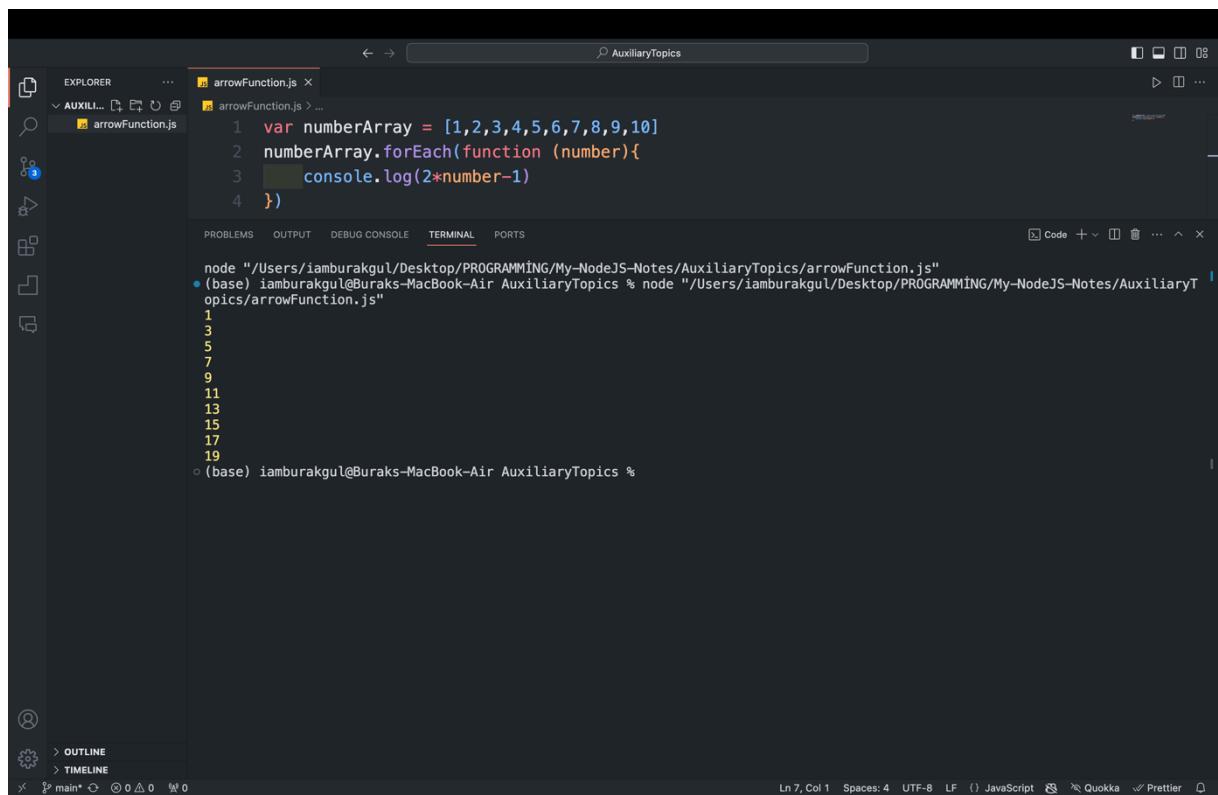
```
AuxiliaryTopics - arrowFunction.js

var numberArray = [1,2,3,4,5,6,7,8,9,10]
```

Bu dizi üzerindeki her eleman için çalışıralım.

```
AuxiliaryTopics - arrowFunction.js

var numberArray = [1,2,3,4,5,6,7,8,9,10]
numberArray.forEach(function (number){
    console.log(2*number-1)
})
```



The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows a folder named "AUXILIARYTOPICS" containing "arrowFunction.js".
- Terminal:** Displays the command "node "/Users/iamburakgul/Desktop/PROGRAMMING/My-NodeJS-Notes/AuxiliaryTopics/arrowFunction.js"" and its output:

```
1
3
5
7
9
11
13
15
17
19
```
- Status Bar:** Shows file information: "Ln 7, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Quokka Quokka Prettier".

Gördüğümüz gibi çalışıyor fakat fonksiyonun içinde function yazmak vs uzatıyor işi onun yerine şunu kullanıksak daha iyi olur gibi.

```
AuxiliaryTopics - arrowFunction.js

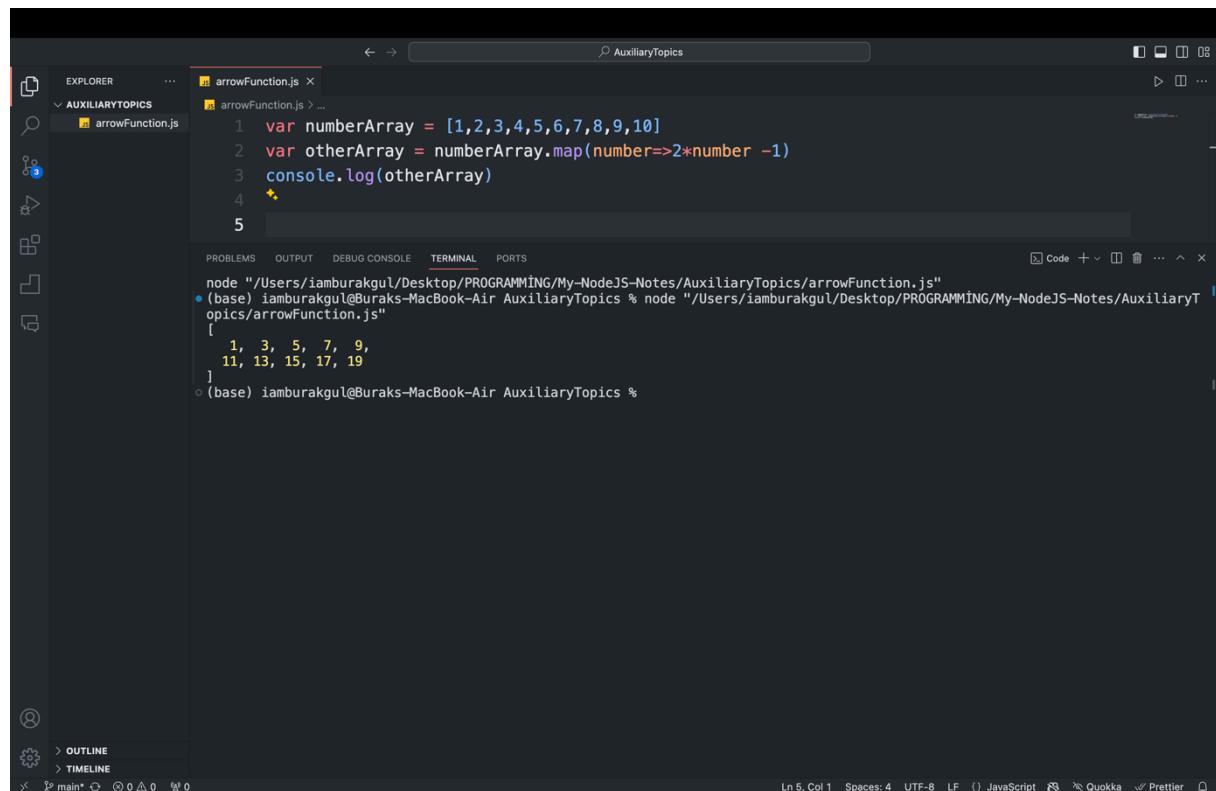
var numberArray = [1,2,3,4,5,6,7,8,9,10]
numberArray.forEach(number => console.log(2*number-1))
```

Yada biz bu 2 katını alıp 1 çıkardığımız diziyi bi yerde kullanacaksak map fonksiyonunu kullanalım.

map fonksiyonu bir fonksiyonu parametre olarak almaktadır ve bu aldığı fonksiyon arrayin her bir elemanı üzerinde işlem yapar ve geri döndürür o elemani. Bu elemanların bir diziye koymak geri döndürme işlemini yapar map fonksiyonu.

```
AuxiliaryTopics - arrowFunction.js

var numberArray = [1,2,3,4,5,6,7,8,9,10]
var otherArray = numberArray.map(number=>2*number -1)
console.log(otherArray)
```



```
EXPLORER AUXILIARYTOPICS arrowFunction.js
var numberArray = [1,2,3,4,5,6,7,8,9,10]
var otherArray = numberArray.map(number=>2*number -1)
console.log(otherArray)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
node "/Users/iamburakgul/Desktop/PROGRAMMING/My-NodeJS-Notes/AuxiliaryTopics/arrowFunction.js"
(base) iamburakgul@Buraks-MacBook-Air AuxiliaryTopics % node "/Users/iamburakgul/Desktop/PROGRAMMING/My-NodeJS-Notes/AuxiliaryTopics/arrowFunction.js"
[
  1, 3, 5, 7, 9,
  11, 13, 15, 17, 19
]
(base) iamburakgul@Buraks-MacBook-Air AuxiliaryTopics %
```

Veya filtreleme işlemi yapacaksak filter fonksiyonunu kullanalım.

Filter fonksiyonu bir array üzerindeki bir elemanın geri döndürülecek diziye dahil edilip edilmeyeceğini belirleyen fonksiyonu parametre almaktadır.

```
AuxiliaryTopics - arrowFunction.js

var numberArray = [1,2,3,4,5,6,7,8,9,10]

var evenNumbers = numberArray.filter(function(number){
    if (number %2 == 0) {
        return true
    }else{
        return false
    }
})
console.log(evenNumbers)

var oddNumbers = numberArray.filter(number=> number%2==1)
console.log(oddNumbers)
```

İşte sonuç.

```
[ 2, 4, 6, 8, 10 ]
[ 1, 3, 5, 7, 9 ]
```

Arrow Functionu seçme sebeplerimizden bir diğeri ise normal functionlar kendi bağlamını oluştururken arrow function kendi bağlamını oluşturmaz.

Normal functionlar çağrıldıkları zaman kendi objectini belirtirken arrow functionlar çağrıldıkları yerdeki objecti belirtiler.

AuxiliaryTopics - arrowFunction.js

```
const person = {  
    name: "Batuhan",  
    lastName: "Gül",  
    hobbies: ["Ride a motorBike", "Travel"],  
    showLastName() {  
        console.log(this.lastName + " is lastName")  
    },  
}  
  
const person2 = {  
    name: "Burak",  
    lastName: "Gül",  
    hobbies: ["Read a Book", "Write a code"],  
    showLastName: () => {  
        console.log(this.lastName + " is lastName")  
    },  
}  
  
person.showLastName(); // Gül is lastName  
person2.showLastName(); // undefined is lastName
```

person nesnesinin showLastName fonksiyonu normal function olduğu için this ile person nesnesini belirtir ve düzgünce çalışır.

person2 nesnesinin showLastName fonksiyonu arrow function olduğu için this ile person2.showLastName() in yazıldığı bağlamı belirtir buradaki this ifadesi o yüzden tanımlı olmadığı için undefined verir.

AuxiliaryTopics - arrowFunction.js

```
const person = {
    name: "Batuhan",
    lastName: "Gül",
    hobbies: ["Ride a motorBike", "Travel"],
    showLastName() {
        console.log(this.lastName + " is lastName ")
    },
    showHobbies() {
        this.hobbies.forEach(function (hobby) {
            console.log(this.name + 's hobbies is ' + hobby)
        })
    }
}
const person2 = {
    name: "Burak",
    lastName: "Gül",
    hobbies: ["Read a Book", "Write a code"],
    showLastName: () => {
        console.log(this.lastName + " is lastName ")
    },
    showHobbies() {
        const self = this
        this.hobbies.forEach(function (hobby) {
            console.log(self.name + 's hobbies is ' + hobby)
        })
    }
}
person.showHobbies()
// undefined 's hobbies is Ride a motorBike,
// undefined 's hobbies is Travel
person2.showHobbies()
// Burak 's hobbies is Read a Book
// Burak 's hobbies is Write a code
```

Burada ise showHobbies fonksiyonlarını anlamaya çalışalım.

person nesnesindeki showHobbies fonksiyonu normal functiondur ve fonksiyon içindeki ilk this person nesnesini belirtir ama forEach içinde ki normal fonksiyon olduğu için yeni bir bağlam oluşturur.O bağlamda this i bulamadığı için undefined yazıyor.

person2 nesnesindeki showHobbies fonksiyonu normal functiondur ve fonksiyon içindeki ilk this person2 nesnesini belirtir ve bunu forEach içinde kullanabilelim diye bir değişkene atayalım ve forEach içindeki normal function kendi bağlamını oluşturur ve self ile person2 nesnesine erişiriz.Burada self in python'daki gibi bir kullanımı söz konusu değildir bir değişken oluşturduk sadece.

```
AuxiliaryTopics - arrowFunction.js

const person = {
  name: "Batuhan",
  lastName: "Gül",
  hobbies: ["Ride a motorBike", "Travel"],
  citties: ["Sivas", "İstanbul", "Bursa"],

  showLastName() {
    console.log(this.lastName + " is lastName ");
  },
  showHobbies() {
    this.hobbies.forEach(function (hobby) {
      console.log(this.name + 's hobbies is ' + hobby);
    });
  },
};

const person2 = {
  name: "Burak",
  lastName: "Gül",
  hobbies: ["Read a Book", "Write a code"],
  citties: ["sivas", "istanbul", "bursa"],

  showLastName: () => {
    console.log(this.lastName + " is lastName ");
  },
  showHobbies() {
    this.hobbies.forEach((hobby) =>
      console.log(this.name + 's hobbies is ' + hobby)
    );
  },
};

person.showHobbies();
// undefined 's hobbies is Ride a motorBike,
// undefined 's hobbies is Travel
person2.showHobbies();
// Burak 's hobbies is Read a Book
// Burak 's hobbies is Write a code
```

person2 deki showHobbies fonksiyonunu bu şekilde yapmayalım diye arrow functionlar var aslında.

```
My-NodeJS-Notes - arrowFunction.js

const person = {
  name: "Batuhan",
  lastName: "Gül",
  hobbies: ["Ride a motorBike", "Travel"],
  showLastName() {
    console.log(this.lastName + " is lastName")
  },
  showHobbies() {
    this.hobbies.forEach(function (hobby) {
      console.log(this.name + "'s hobbies is " + hobby)
    })
  }
}
const person2 = {
  name: "Burak",
  lastName: "Gül",
  hobbies: ["Read a Book", "Write a code"],
  showLastName: () => {
    console.log(this.lastName + " is lastName")
  },
  showHobbies() {
    const self = this
    this.hobbies.forEach(function (hobby) {
      console.log(self.name + "'s hobbies is " + hobby)
    })
  },
  showHobbies2(){
    this.hobbies.forEach(hobby=>console.log(this.name + "'s hobbies is " + hobby ))
  }
}
person.showHobbies()
// undefined 's hobbies is Ride a motorBike,
// undefined 's hobbies is Travel
person2.showHobbies()
// Burak 's hobbies is Read a Book
// Burak 's hobbies is Write a code
person2.showHobbies2()
// Burak 's hobbies is Read a Book
// Burak 's hobbies is Write a code
```

forEach içinde arrow function kullanarak nesnemize rahatça erişik yeni bir değişkene atamadan.

Şimdi ise geçen hafta kaldığımız yerden devam edelim.

Section2-3 ü Section2-3-4 olarak değiştirelim.

index.js içindeki fonksiyonlardan arrow function'a çevirebileceklerimizi çevirelim.

Öncelikle args içindeki handler functionundan function keywordünü kaldırıralım

```
My-NodeJS-Notes - index.js
handler(argv) {
  notes.addNote(argv.title, argv.body)
}
```

Fonksiyonumuz bu hale gelecektir daha sonra add ,remove,list ve read içinde aynısını yapalım.

Şimdi de notes.js dosyasına bakalım.

addNote fonksiyonu içindeki filter fonksiyonu

```
My-NodeJS-Notes - notes.js
const duplicateNotes = notes.filter(function (note) {
  return note.title === title
})
```

bu halden

```
My-NodeJS-Notes - notes.js
const duplicateNotes = notes.filter((note) => note.title === title)
```

Bu hale gelecektir arrow function sayesinde.

removeNote kısmındaki filter fonksiyonunun birinci ve ikinci durumu şu şekilde olacaktır.

```
My-NodeJS-Notes - notes.js
// * First state
const notesToKeep = notes.filter(function (note) {
  return note.title !== title
})
// * Second state
const notesToKeep = notes.filter((note) => note.title !== title)
```

saveNotes ise şu şekilde olacaktır :

```
My-NodeJS-Notes - notes.js

// * First state
const saveNotes = function (notes) {
  const dataJSON = JSON.stringify(notes)
  fs.writeFileSync("notes.json", dataJSON)
}

// * Second state
const saveNotes = (notes) => {
  const dataJSON = JSON.stringify(notes)
  fs.writeFileSync("notes.json", dataJSON)
}
```

loadNotes ise şu şekilde:

```
My-NodeJS-Notes - notes.js

/* First State
function loadNotes() {
  try {
    // read from file
    const dataBuffer = fs.readFileSync("notes.json")
    // Convert to JSON format
    const dataJSON = dataBuffer.toString()
    // Parse the string and return
    return JSON.parse(dataJSON)
  } catch (error) {
    return []
  }
}

/* Second State
const loadNotes = () => {
  try {
    // read from file
    const dataBuffer = fs.readFileSync("notes.json")
    // Convert to JSON format
    const dataJSON = dataBuffer.toString()
    // Parse the string and return
    return JSON.parse(dataJSON)
  } catch (error) {
    return []
  }
}
```

Şimdi ise command satırında listeleme yapmak için notes.js dosyasında bir fonksiyon oluşturalım listNotes adında bunu notes.js dosyasından export edip index.js de list command yargında kullanalım.

```
My-NodeJS-Notes - notes.js

const listNotes = () => {console.log("listNotes Function");}

module.exports = {
  addNote: addNote, // ! sağ kısımda yazan bu dosyadaki ile eşleşmeli
  removeNote: removeNote ,// ! sol kısımda yazan ise başka dosyalarda kullanılmak üzere isimlendirilir.
  listNotes : listNotes
}
```

```
My-NodeJS-Notes - index.js

yargs.command({
  command: "list",
  describe: "List your notes",
  handler() {
    notes.listNotes()
  }
})
```

Şimdi deneyelim bakalım çalışacak mı .

```
notes.js index.js
33 },
34 handler(argv) {
35   notes.removeNote(argv.title)
36 }
37 )
38 yargs.command({
39   command: "list",
40   describe: "List your notes",
41   handler() {
42     notes.listNotes()
43   }
44 })
45 yargs.command({
46   command: "read",
47   describe: "Read a note",
48 })
49 
```

(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js list
listNotes Function
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 %

list commandindeki notes.listNotes() fonksiyonu çalıştı ve bu fonksiyonu arrow function olarak yazdık.

listNotes fonksiyonunu güncelleyelim

```
const listNotes = () => {
  const notes = loadNotes()
  console.log(chalk.inverse("Your notes"))
  notes.forEach((note) => {
    console.log(note.title)
  })
}
```

Şimdi ise yeniden çalıştıralım.

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "MY-NODEJS-NOTES".
- Editor View:** Displays the code for "notes.js".

```
const saveNotes = (notes) => {
  const dataJSON = JSON.stringify(notes)
  fs.writeFileSync("notes.json", dataJSON)
}

const listNotes = () => {
  const notes = loadNotes()
  console.log(chalk.inverse("Your notes"))
  notes.forEach((note) => {
    console.log(note.title)
  })
}
```
- Terminal View:** Shows the command "node index.js list" being run and its output:

```
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js list
Your notes
Burak
Batuhan
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 %
```
- Status Bar:** Shows the current file is "notes.js", line 76, column 28, and the file is saved.

Gördüğümüz üzere listeleme işlemlerini de tamamladık.

Şimdi ise note eklerken notlarda var mı yok mu diye filter methodunu kullanarak eklüyoruz. Filter methodu arrayin tüm elemanlarını gezeceği için big o notasyonu O(n) olacaktır ama find methodunu kullanırsak tüm diziyi gezmemize gerek kalmayabilir.

Ne addNote fonksiyonu şu hale gelmektedir.

```
My-NodeJS-Notes - notes.js

// * First state
function addNote(title, body) {
  const notes = loadNotes()
  const duplicateNotes = notes.filter((note) => note.title === title)
  if (duplicateNotes.length === 0) {
    // ! no duplicate
    notes.push({ title: title, body: body })
    saveNotes(notes)
    console.log(chalk.green.inverse("New note added!"))
  } else {
    // ! there is duplicate
    console.log(chalk.red.inverse("Note title taken"))
  }
}

// * Second State
function addNote(title, body) {
  const notes = loadNotes()
  const duplicateNote = notes.find(note=>note.title === title)
  if (duplicateNote==>undefined) {
    // ! no duplicate
    notes.push({ title: title, body: body })
    saveNotes(notes)
    console.log(chalk.green.inverse("New note added!"))
  } else {
    // ! there is duplicate
    console.log(chalk.red.inverse("Note title taken"))
  }
}
```

Çalışıp çalışmadığını görmek için önce listeleme işlemış yapıp daha sonra yeni veri ekleyip bir daha listeleme işlemi yapacağız.

The screenshot shows a VS Code interface with the following details:

- Explorer View:** Shows a project structure named "MY-NODEJS-NOTES" containing "AuxiliaryTopics", "node_modules", "Section1", "Section2-3-4", and "notes.js".
- Editor View:** Two tabs are open: "notes.js" and "index.js". The "notes.js" tab contains code for reading and writing notes using fs and chalk modules.
- Terminal View:** The terminal shows the following interaction:

```
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js list
Your notes
Burak
Batuhan
• (base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js add --title="BTU" --body="BM"
New note added!
• (base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js list
Your notes
Burak
Batuhan
BTU
◦ (base) iamburakgul@Buraks-MacBook-Air Section2-3-4 %
```
- Status Bar:** Shows file paths, line count (Ln 17), character count (Col 5), spaces (Spaces: 2), encoding (UTF-8), and line endings (LF).

Gördüğümüz üzere çalışıyor bir sorun yok.

Şimdi ise hatırlarsanız title benzersiz olacaktı ve bu benzersiz olmasından ötürü ekleme ve silme listeleme işlemlerimiz daha rahat olacaktı.

Şimdi ise bir notu okuma işlemi için index.js deki read yarg.commandını şu hale getirelim :

```
My-NodeJS-Notes - index.js

yargs.command({
  command: "read",
  describe: "Read a note",
  builder: {
    title: {
      describe: "Note title",
      demandOption: true,
      type: "string"
    }
  },
  handler() {
    console.log("Reading a note!")
  }
})
```

Burada kullanmak üzere notes.js içinde bir method oluşturalım ve export edelim.

```
My-NodeJS-Notes - notes.js

const readNote = (title) => {
  const notes = loadNotes()
  const note = notes.find((note) => note.title === title)
  if (note) {
    console.log(chalk.inverse(note.title))
    console.log(note.body)
  } else {
    console.log(chalk.red.inverse("Note not found!"))
  }
}

module.exports = {
  addNote: addNote, // ! sağ kısımda yazan bu dosyadaki ile eşleşmeli
  removeNote: removeNote, // ! sol kısımda yazan ise başka dosyalarda kullanılmak üzere isimlendirilir.
  listNotes: listNotes,
  readNote:readNote
}
```

Not bulunursa notu ve body sini yazdırıyoruz bulamazsa bulamadık diyoruz.

Şimdi bu readNote fonksiyonunu yarg commandin handlerına ayarlayıp kullanalım.

```
My-NodeJS-Notes - index.js

yargs.command({
  command: "read",
  describe: "Read a note",
  builder: {
    title: {
      describe: "Note title",
      demandOption: true,
      type: "string"
    }
  },
  handler(argv) {
    notes.readNote(argv.title)
  }
})
```

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "MY-NODEJS-NOTES".
- Editor View:** Displays the file "index.js" containing Node.js code using the `yargs` module to handle command-line arguments for listing and reading notes.
- Terminal View:** Shows the terminal output where the application is run and demonstrates its functionality for listing and reading notes.

```
index.js
Section2-3-4 > index.js > ...
36     }
37   })
38   yargs.command({
39     command: "list",
40     describe: "List your notes",
41     handler() {
42       notes.listNotes()
43     }
44   })
45   yargs.command({
46     command: "read",
47     describe: "Read a note",
48     builder: {
49       title: {
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js list
Your notes
Burak
Batuhan
BTU
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 % node index.js read --title="BTU"
BTU
BM
(base) iamburakgul@Buraks-MacBook-Air Section2-3-4 %
```

Gördüğümüz üzere okuma işlemi de tamamdır.