Getting Input from User
Run node app.js Can
How can we access this argument?

console.log(process.argv) (argument vector - an array)
Rerun the script. We have a new array. Three elements: Path for the node executable. Path for the file. And the argument

Now run
console.log(process.argv[2]) we just get the argument we want

We will use add, remove, list etc as the argument

Remove all the codes but the two imports for chalk and notes

const chalk ..
const getNotes ..

const command = process.argv[2]

```
if (command === 'add')
{
 Console.log('Adding note!')
}
```

We can also add remove

```
if (command === 'add')
{
 console.log('Adding note!')
}  else if (command === 'remove')
{
 console.log('Removing note!')
}
```

Now we need title and content.

Run node app.js add —title="This is my title"

Add console.log(process.argv) to dump the array.

The fourth argument is the title but it is not parsed. We need to check if the title is provided, etc. But for testing and maintenance, it is best to find a related npm module to parse arguments.

We are going to use yards package. Install the package.
npm i yargs

Add the import
const yargs = require('yargs')

Remove if conditions below the log dump.
Add console.log(yargs.argv)

Run node app.js and compare the outputs of two.

Now run node app.js add —title="Shopping list"

Now run node app.js —help
Options for yargs

Run node app.js —version
In the code, add..
//Customize yargs version
yargs.version('1.1.0') and run the code again.

```
// Create add command
yargs.command( {
  command: 'add',
  describe: 'Add a new note',
  handler: function () {
    console.log('Adding a new note!')
  }
})
```

Now run node app.js —help

Now run node app.js add

Now add remove command

```
// Create remove command
yargs.command( {
  command: 'remove',
  describe: 'Remove a new note',
  handler: function () {
    console.log('Removing the note!')
  }
})
```

Now run node app.js —help

Now run node app.js —remove

Now add list command

```
// Create list command
yargs.command( {
  command: 'list',
  describe: 'List your notes',
  handler: function () {
    console.log('Listing all notes!')
  }
})
```

Now add read command

```
// Create read command
yargs.command( {
  command: 'read',
  describe: 'Read a note',
  handler: function () {
    console.log('Reading a note!')
  }
})
```

Test now!
Now run node app.js —help

Now run node app.js —listIread

Now add builders:

```
// Create add command
yargs.command( {
  command: ‘add’,
  describe: ‘Add a new note’,
  builder: {
    title: {
      describe: ’Note title’
    }
  },
  handler: function (argv) {
    console.log(‘Adding a new note!’, argv)
  }
})
```

Now run
node app.js add —title=“Shopping list”
But title is not required. We can also run
node app.js add

Now make it required.

```
yargs.command( {
  command: ‘add’,
  describe: ‘Add a new note’,
  builder: {
    title: {
      describe: ’Note title’,
      demandOption: true
    }
  },
  handler: function (argv) {
    console.log(‘Adding a new note!’, argv)
  }
})
```

Also require a string
Try running
node app.js add —title

Title is true now. Because it is considered boolean when not a string.

Now enforce string type

```
yargs.command( {
  command: ‘add’,
  describe: ‘Add a new note’,
  builder: {
    title: {
      describe: ’Note title’,
      demandOption: true,
      type: ’string’
    }
  },
```

```
  handler: function (argv) {
    console.log('Adding a new note!', argv)
  }
})
```

Try running
node app.js add —title

Same result but empty string now.

Now remove console.log(yargs.argv) at the end of the code.
No output will be shown. We need a parser now.

yargs.parse()

Now we will only print the title. Remove console.log from the handler and put console.log('Title: ', + argv.title) title matches the title in the builder.

Add a body option for the add command.

```
yargs.command( {
  command: 'add',
  describe: 'Add a new note',
  builder: {
    title: {
      describe: 'Note title',
      demandOption: true,
      type: 'string'
    },
    body: {
      describe: 'Note body',
      demandOption: true,
      type: 'string'
    }
  },
  handler: function (argv) {
    console.log('Title: ', argv.title)
    console.log('Body: ', argv.body)
  }
})
```

Now run node app.js add —title="Buy"

Fails. Requires body.
Rerun run node app.js add —title="Buy" —body="these things.."

Storing data with json
Each object will represent a note. Json has native support in JS.

Close all folders and collapse notes-app folder.
Create a new folder for playground.
Create a new script: 1-json.js

```
const book = {
  title: 'my title',
```

```
  author: 'can conomo'
}

Take a string and create json representation.
const bookJSON = JSON.stringify(book)
console.log(bookJSON)

Go to cd ../playground
node 1-json.js

bookJSON is string. Title property doesn't exist.
Try console.log(bookJSON.title) will fail.

const bookJSON = JSON.stringify(book)
console.log(bookJSON)

Const parsedObject = JSON.pase(bookJSON) // or array.
console.log(parsedObject.title)

Remove logs and it will be like this:
const fs = require('fs')

const book = {
  title: 'my title',
  author: 'can conomo'
}

const bookJSON = JSON.stringify(book)
Fs.writeFileSync('1-json.json', bookJson)

Let's try loading the file. Comment all but the fs line.

const fs = require('fs')
const dataBuffer = fs.readFileSync('1-json.json')
console.log(dataBuffer)

See the output.
Try console.log(dataBuffer.toString())

Remove it. Now it looks like this:

const fs = require('fs')

const dataBuffer = fs.readFileSync('1-json.json')
const dataJSON = dataBuffer.toString()
const data = JSON.parse(dataJSON)
console.log(data.title)
```