JavaScript

Boitumelo Phetla[1]

[1] *PluralSight Google ScholarShip*

June 16, 2019

JavaScript, is a lightweight interpreted or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.

1 Simplistic JavaScript 1

1.1 Command-line based programming

A simple project:

```
$bash: touch {index.html,script.js,style.css}
$bash: tree
   __________ index.html
   __________ script.js
   __________ style.css
```

Include the script (**javascript**) and the page styling script (**cascading stylesheet**) files into the *index.html*.

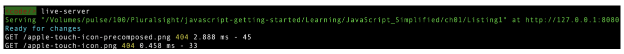
```
<!DOCTYPE>
   <html>
      <head>
            <script src="path/*.js"></script>
            <link rel="stylesheet"
                 href="path/*.css">
      </head>
            <body>
                  <div>
                        <header></header>
                  </div>
                     <div><!-- body --></div>
                  <div>
                     <footer></footer>
                  </div>
            </body>
   </html>
```

Add some simple HTML markup code and launch a live-server of the code.

```
<!DOCTYPE>
<html>
      <head>
            <script src="script.js"></script>
            <link rel="stylesheet"
                 href="style.css">
      </head>
            <body>
                  <div id="header">
                        <h1>Welcome to
                             JavaScript</h1>
                  </div>
            </body>
</html>
```

Launch the command-line (Terminal)

```
$bash: live-server
```

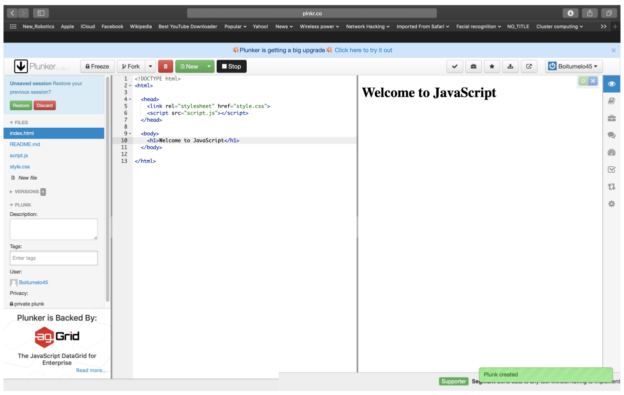


```
<code/> live-server
Serving "/Volumes/pulse/100/Pluralsight/javascript-getting-started/Learning/JavaScript_Simplified/ch01/Listing1" at http://127.0.0.1:8080
Ready for changes
GET /apple-touch-icon-precomposed.png 404 2.888 ms - 45
GET /apple-touch-icon.png 404 0.458 ms - 33
```

Figure 1: *Live-server*

1.2 Plunker

Or create an account on Plunker. Plunker sets up your working environment for you.



```
<!DOCTYPE html>
<html>

  <head>
    <link rel="stylesheet" href="style.css">
    <script src="script.js"></script>
  </head>

  <body>
    <h1>Welcome to JavaScript</h1>
  </body>

</html>
```

Welcome to JavaScript

Figure 2: *Plunker*

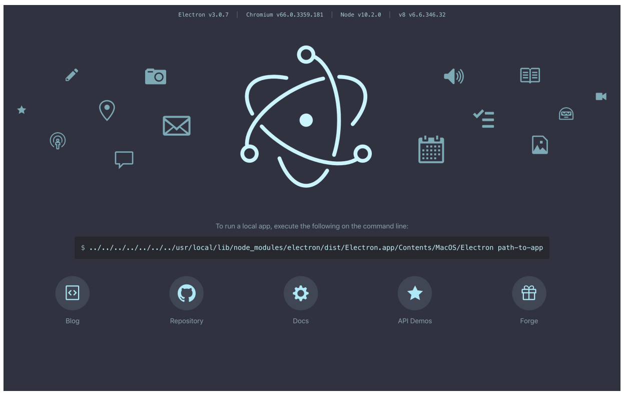
1.3 Electron

Watch this video Electron.

```
# Clone the Quick Start repository
$ git clone
    https://github.com/electron/electron-quick-start

# Go into the repository
$ cd electron-quick-start

# Install the dependencies and run
$ npm install && npm start
```



To run a local app, execute the following on the command line:

```
$ ../../../../../../usr/local/lib/node_modules/electron/dist/Electron.app/Contents/MacOS/Electron path-to-app
```

Electron v3.0.7 | Chromium v66.0.3359.181 | Node v10.2.0 | v8 v6.6.346.32
Blog
Repository
Docs
API Demos
Forge

Figure 3: *Electron*

```
$bash: mkdir Electron1; cd Electron1; npm init
{
  "name": "electron1",
  "version": "1.0.0",
  "description": "First App",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Electron"
  ],
  "author": "Boitumelo Phetla",
  "license": "ISC"
}
```

At this point, you'll need to install electron itself. The recommended way of doing so is to install it as a development dependency in your app, which allows you to work on multiple apps with different Electron versions. To do so, run the following command from your app's directory:

```
$bash: npm install --save-dev electron
$bash: tree -L 1
        .
        |____________node_modules
        |____________package-lock.json
        |____________package.json

1 directory, 2 files
```

All APIs and features found in Electron are accessible through the electron module, which can be required like any other Node.js module:

```
const electron = require('electron')
```

To avoid any huddles, try this simple example.

```
# Clone the repository
$ git clone
    https://github.com/electron/electron-quick-start
# Go into the repository
$ cd electron-quick-start
# Install dependencies
$ npm install
# Run the app
$ npm start
```

1.4 Meteor

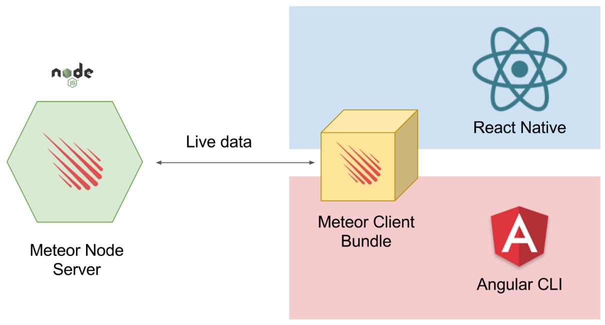


Figure 4: *Meteor*

To create the app, open your terminal and type:

```
$bash: meteor create simple-todos
```

```
output:

Created a new Meteor app in 'simple-todos'.

To run your new app:
  cd simple-todos
  meteor

If you are new to Meteor, try some of the
    learning resources here:
  https://www.meteor.com/tutorials

To start with a different app template, try one
    of the following:

  meteor create --bare  # to create an empty app
  meteor create --minimal # to create an app
      with as few Meteor packages as possible
  meteor create --full  # to create a more
      complete scaffolded app
```

1.5 Coding in JavaScript

1.5.1 Variables

```
"use strict";
//let is accessible in the code block where it
    is used
let firstName = "John Doe"; //camelCasing
console.log(firstName);

/*Output*/
$bash: node let.js
John Doe
```

1.5.2 Global variable, function, Operators

```
"use strict";

//A = P(1 + rt)
let r = 10.5, t = 5, p = 200;

var A = (r,t,p) => {
  return p*(1 + (r/100)*t);
}

let interest = A(r,t,p);
console.log("R200 (interest in 5 years at at
    interest rate of 10.5% = R" + interest +
    "-00)");
```

1.5.3 Simple function

```
"use strict";

//A = P(1 + rt)
let r = 10.5, t = 5, p = 200;

//function definition (without using arrow
    function)
var A = function(r,t,p){
    return p*(1 + (r/100)*t);
}

console.log(A(r,t,p));
```

1.5.4 Variables and block code

```
"use strict";

array = [1,2,3,4,5];
var count = 0;
let counter = 0;

for(let i = 0; i < array.length; i++){
    count += array[i];
    counter += i;
    if(count > 5){
        var num1 = count*5;  //accessible
            outside this code block
        let num2 = num1*5;   //only accessible
            within this code block
        console.log("num1: ", num1, ',', 'num2:
            ', num2);
    }
    //console.log("xnum1: ", num1, ',', 'xnum2:
        ', num2);
    try{
        console.log("xnum1: ", num1, ',',
            'xnum2: ', num2);
    }catch{
        console.log('xnum1: ', num1); //<--
            accessing num1
        console.log('xnum2: ', 'This will not
            print because it is not
            accessible'); //<-- can't access num2
    }

}
console.log('count: ', count, ',' , 'counter:
    ',counter); //<-- both count and counter
    are accessible because they are in the same
    code block

/*Output*/
xnum1: undefined
xnum2: This will not print because it is not
    accessible
xnum1: undefined
xnum2: This will not print because it is not
    accessible
num1: 30 , num2: 150
xnum1: 30
xnum2: This will not print because it is not
    accessible
num1: 50 , num2: 250
xnum1: 50
xnum2: This will not print because it is not
    accessible
```

```
num1: 75 , num2: 375
xnum1: 75
xnum2: This will not print because it is not
    accessible
count: 15 , counter: 10
```

1.5.5 Type of primitive data

```
"use strict";

let b = false;
let array = [1,2,3,'hello', 3.02, b];

var typeOfData = (array) =>{
  array.forEach((element) =>{
    console.log(element, 'is a ',
        typeof(element));
  })
}

typeOfData(array);

/*Output*/

1 'is a ' 'number'
2 'is a ' 'number'
3 'is a ' 'number'
hello is a string
3.02 'is a ' 'number'
false 'is a ' 'boolean'
```

1.5.6 Undefined and Null

```
"use strict";

let anUndefinedVariable; //not initialized
let empty = null; //is empty (nothing)

console.log(anUndefinedVariable, empty);
console.log(typeof(anUndefinedVariable),
    typeof(empty));

/*Output*/

undefined null
undefined object
```

1.5.7 Data containers

Array

```
"use strict";

/*
  We use arrays to contain multiple variables
      values instead of declaring a thousand of
      them.
*/

let array = ["John", "Doe", 34, "X", "USA",
    "Nevada", "Porsche 911", ["soccer",
    "volleyball", "chess"], ["python", "nim",
    "c", "java", "julia", "objective C", "SQL",
    "GraphQL", "JavaScript", "HTML5", "CSS3",
    "jQuery", "Machine Learning",
    "Bash"],"MIT", "In a relationship",
    ["Bali", "Singapore", "Hong Kong",
    "Thailand", "Mozambique", "Swaziland",
    "South Africa", "Lombark"], ["Electrical",
    "Computer"]];

array.forEach((element)=>{console.log(element)});

/*Output*/

John
Doe
34
X
USA
Nevada
Porsche 911
[ 'soccer', 'volleyball', 'chess' ]
[ 'python',
  'nim',
  'c',
  'java',
  'julia',
  'objective C',
  'SQL',
  'GraphQL',
  'JavaScript',
  'HTML5',
  'CSS3',
  'jQuery',
  'Machine Learning',
  'Bash' ]
MIT
In a relationship
[ 'Bali',
  'Singapore',
  'Hong Kong',
  'Thailand',
  'Mozambique',
  'Swaziland',
  'South Africa',
  'Lombark' ]
[ 'Electrical', 'Computer' ]
```

Add values into an empty array

```
"use strict";

/*
  We use arrays to contain multiple variables
      values instead of declaring a thousand of
      them.
*/

let array = ["John", "Doe", 34, "X", "USA",
    "Nevada", "Porsche 911"];
let results = [] //empty array
```

```
/*
  add elements into array
  array.push(value)
*/

for(let i = 0; i < array.length; i++){
    results.push(array[i]);
}

console.log("Length of results[]: ",
     results.length);
console.log(results);

/*Output*/
Length of results[]: 7
[ 'John', 'Doe', 34, 'X', 'USA', 'Nevada',
    'Porsche 911' ]
```

Removing elements from an array

```
"use strict";

let array = ["John", "Doe", 34, "X", "USA",
    "Nevada", "Porsche 911"];
/*
 remove elements from an array
 array.pop(); //removes last value
*/
while(array.length > 0){
        array.pop();
}

console.log("Length of array: ", array.length);
console.log(array);

/*Output*/
Length of array: 0
[]
```

Removing the first elements by shifting the array.

```
"use strict";

let array = ["John", "Doe", 34, "X", "USA",
    "Nevada", "Porsche 911"];
array.shift(); //shifts the array

console.log(array);

/*Output*/

[ 'Doe', 34, 'X', 'USA', 'Nevada', 'Porsche 911'
    ]
```

Deleting elements from an array.

```
"use strict";

let array = ["cobol", "c#", ".NET", "Python"];

/*
  delete the first three elements of the array
*/

let languages_depricated = array.splice(0,3);
console.log(array, languages_depricated);

/*Output*/
[ 'Python' ] [ 'cobol', 'c#', '.NET' ]
```

Deleting elements from an array and mutating it.

```
"use strict";

let array = ["cobol", "c#", ".NET", "Python"];

/*
  delete the first three elements of the array
      and mutating the array
  using splice()

   splice(0,3) - means:
   delete element from index 0 and delete 3 items
   if array = [1,2,3,4]
   splice(0,3)
        performs:
                    [2,3,4] - 1 : delete[1]
                    [3,4] - 2 : delete[2]
                    [4] - 3 : delete[3]
                    all at index 0
        returns new array = [4]
*/

let deleted = array.splice(0,3, "Java", "C",
    "Nim", "Objective C", "Swing");
console.log(array, deleted);

/*Output*/
[ 'Java', 'C', 'Nim', 'Objective C', 'Swing',
    'Python' ] [ 'cobol', 'c#', '.NET' ]
```

Some of Array methods you can use.

forEach() This method can help you to loop over array's items.

```
const arr = [1, 2, 3, 4, 5, 6];

  arr.forEach(item => {
    console.log(item); // output: 1 2 3 4 5 6
  });
```

includes() This method check if array includes the item passed in the method.

```
const arr = [1, 2, 3, 4, 5, 6];

 arr.includes(2); // output: true
 arr.includes(7); // output: false
```

filter() This method create new array with only elements passed condition inside the provided function.

```
const arr = [1, 2, 3, 4, 5, 6];

  // item(s) greater than 3
```

```
const filtered = arr.filter(num => num >
    3);
console.log(filtered); // output: [4, 5, 6]

console.log(arr); // output: [1, 2, 3, 4,
    5, 6]
```

map() This method create new array by calling the provided function in every element.The reduce() method applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value - MDN

```
const arr = [1, 2, 3, 4, 5, 6];

 // add one to every element
 const oneAdded = arr.map(num => num + 1);
 console.log(oneAdded); // output [2, 3, 4,
     5, 6, 7]

 console.log(arr); // output: [1, 2, 3, 4,
     5, 6]
```

reduce() This method check if at least one of array's item passed the condition. If passed, it return 'true' otherwise 'false'.

```
const arr = [1, 2, 3, 4, 5, 6];

 const sum = arr.reduce((total, value) =>
     total + value, 0);
 console.log(sum); // 21
```

some() This method check if at least one of array's item passed the condition. If passed, it return 'true' otherwise 'false'.

```
const arr = [1, 2, 3, 4, 5, 6];

// at least one element is greater than 4?
const largeNum = arr.some(num => num > 4);
console.log(largeNum); // output: true

// at least one element is less than or
    equal to 0?
const smallNum = arr.some(num => num <= 0);
console.log(smallNum); // output: false
```

every() This method check if all array's item passed the condition. If passed, it return 'true' otherwise 'false'.

```
const arr = [1, 2, 3, 4, 5, 6];

  // all elements are greater than 4
  const greaterFour = arr.every(num => num >
      4);
  console.log(greaterFour); // output: false

  // all elements are less than 10
  const lessTen = arr.every(num => num < 10);
  console.log(lessTen); // output: true
```

sort() This method used to arrange/sort array's item either ascending or descending order.

```
const arr = [1, 2, 3, 4, 5, 6];
const alpha = ['e', 'a', 'c', 'u', 'y'];

// sort in descending order
descOrder = arr.sort((a, b) => a > b ? -1 :
    1);
console.log(descOrder); // output: [6, 5,
    4, 3, 2, 1]

// sort in ascending order
ascOrder = alpha.sort((a, b) => a > b ? 1 :
    -1);
console.log(ascOrder); // output: ['a',
    'c', 'e', 'u', 'y']
```

Array.from() This change all thing that are array-like or iterable into true array especially when working with DOM, so that you can use other array methods like reduce, map, filter and so on.

code 1

```
const name = 'frugence';
const nameArray = Array.from(name);

console.log(name); // output: frugence
console.log(nameArray); // output: ['f',
    'r', 'u', 'g', 'e', 'n', 'c', 'e']
```

code 2

```
// I assume that you have created unorder
    list of items in our html file.

const lis = document.querySelectorAll('li');
const lisArray =
    Array.from(document.querySelectorAll('li'));

// is true array?
console.log(Array.isArray(lis)); // output:
    false
console.log(Array.isArray(lisArray)); //
    output: true
```

Array.of() This create array from every arguments passed into it.

```
const nums = Array.of(1, 2, 3, 4, 5, 6);
console.log(nums); // output: [1, 2, 3, 4,
    5, 6]
```

Dictionary

```
"use strict";

let data = {

    "first name": "John",
    "last name" : "Doe",
    "age"       : 34,
    "company"   : "X",
    "country"   : "USA",
    "State"     : "Nevada",
    "car"       : "Porsche 911",
    "hobby"     : ["soccer", "volleyball",
        "chess"],
    "polyglot" : ["python", "nim", "c", "java",
        "julia", "objective C", "SQL",
        "GraphQL", "JavaScript", "HTML5",
        "CSS3", "jQuery", "Machine Learning",
        "Bash"],
    "university" : "MIT",
    "status"    : "in a relationship",
    "travels"   : ["Bali", "Singapore", "Hong
        Kong", "Thailand", "Mozambique",
        "Swaziland", "South Africa", "Lombark"],
    "Degrees"   : ["Electrical", "Computer"]


}

console.log(data);


/*Output*/

{ 'first name': 'John',
  'last name': 'Doe',
  age: 34,
  company: 'X',
  country: 'USA',
  State: 'Nevada',
  car: 'Porsche 911',
  hobby: [ 'soccer', 'volleyball', 'chess' ],
  polyglot:
   [ 'python',
     'nim',
     'c',
     'java',
     'julia',
     'objective C',
     'SQL',
     'GraphQL',
     'JavaScript',
     'HTML5',
     'CSS3',
     'jQuery',
     'Machine Learning',
     'Bash' ],
  university: 'MIT',
  status: 'in a relationship',
  travels:
   [ 'Bali',
     'Singapore',
     'Hong Kong',
     'Thailand',
     'Mozambique',
     'Swaziland',
     'South Africa',
     'Lombark' ],
  Degrees: [ 'Electrical', 'Computer' ] }
```

1.5.8 Blackjack project (PluralSight)

```
/*
  Blackjack game of cards
*/

let card1 = "Ace of Spades", card2 = "Ten of
    hearts";
let cards = [card1, card2];

console.log("Welcome to Blackjack");
console.log("You are dealt: ");
cards.forEach((element) => {
    console.log("\t" + element);
})

/*Output*/
Welcome to Blackjack
You are dealt:
        Ace of Spades
        Ten of hearts
```

For loops, Arrays

```
/*
  Blackjack game of cards
*/

let suits = ["Heart", "Clubs", "Diamonds",
    "Spades"];
let values = ["Ace", "King", "Queen", "Jack",
    "Ten", "Nine", "Eight", "Seven", "Six",
    "Five", "Four", "Three", "Two"];


let deck = []

for(let suitIdx = 0; suitIdx < suits.length;
     suitIdx++){
    for(let valueIdx = 0; valueIdx <
         values.length; valueIdx++){
        deck.push(values[valueIdx] + ' of ' +
             suits[suitIdx]);
    }
}

console.log(deck);

/*Output*/
....
 'Four of Spades',
 'Three of Spades',
 'Two of Spades' ]
```

Advancing the Blackjack code.

```
/*
Blackjack game of cards
*/

let suits = ["Heart", "Clubs", "Diamonds",
    "Spades"];
let values = ["Ace", "King", "Queen", "Jack",
    "Ten", "Nine", "Eight", "Seven", "Six",
    "Five", "Four", "Three", "Two"];


function createDeck(){
   let deck = [] //crear deck
   for(let suitIdx = 0; suitIdx < suits.length;
       suitIdx++){
        for(let valueIdx = 0; valueIdx <
            values.length; valueIdx++){
             deck.push(values[valueIdx] + ' of
                 ' + suits[suitIdx]);
        }
   }
        return deck
}

let deck = createDeck()
//console.log(deck);

function getNextCard(){
   return deck.shift()
}


let playerCards = []

for(let i = 0; i < 2; i++){
   playerCards.push(getNextCard())
}

console.log(playerCards);
```

Objects and functions in the code.

```
/*
Blackjack game of cards
*/

let suits = ["Heart", "Clubs", "Diamonds",
    "Spades"];
let values = ["Ace", "King", "Queen", "Jack",
    "Ten", "Nine", "Eight", "Seven", "Six",
    "Five", "Four", "Three", "Two"];


function createDeck(){
   let deck = [] //crear deck
   for(let suitIdx = 0; suitIdx < suits.length;
       suitIdx++){
        for(let valueIdx = 0; valueIdx <
            values.length; valueIdx++){
             deck.push(values[valueIdx] + ' of
                 ' + suits[suitIdx]);
        }
   }
        return deck
}

let deck = createDeck()
//console.log(deck);

function getNextCard(){
   return deck.shift()
}


let playerCards = []

for(let i = 0; i < 2; i++){
   playerCards.push(getNextCard())
}

console.log(playerCards);
```

Objects

```
/*
Blackjack game of cards
*/

let suits = ["Heart", "Clubs", "Diamonds",
    "Spades"];
let values = ["Ace", "King", "Queen", "Jack",
    "Ten", "Nine", "Eight", "Seven", "Six",
    "Five", "Four", "Three", "Two"];


function createDeck(){
   /*
        Creates a deck of 52 cards
   */
   let deck = [] //crear deck
   for(let suitIdx = 0; suitIdx < suits.length;
       suitIdx++){
        for(let valueIdx = 0; valueIdx <
            values.length; valueIdx++){
             let card = {
                       suit : suits[suitIdx],
                       value: values[valueIdx]
             }
             deck.push(card);
        }
   }
        return deck
}

let deck = createDeck()
//console.log(deck);

function getNextCard(){
   /*Moves to the next card from the card on
       top*/
   return deck.shift()
}

var getCardString = (card) =>{
   /*
        takes object { suit: "v1", valueL "v2"}
        returns: v2 of v1
   */
   return card.value + ' of ' + card.suit
}

let playerCards = []

for(let i = 0; i < 2; i++){
   playerCards.push(getCardString(getNextCard()))
}

console.log("Welcome to BlackJack Game!");
console.log("You are dealt: ");
console.log(playerCards);
```

2 Functions

Simple function.

```
function showMessage(){
     console.log("This is a simple function");
}

showMessage() //This is a simple function
```

Passing data into a function.

```
function showMessage(message){
        console.log(message);
}

showMessage("Hello, world!") //Hello, world!
```

Return statement in a function.

```
"use strict"

function doubles(number){
     return number*2;
}

var r = doubles(2)
console.log(r); //4
```

3 Objects

3.1 Create an Object

```
"use strict"

//object person
let person = {
     name : "x",
     surname: "y",
     age: 0,
     occupation: "a",
     vehicle: "b"
}
```

```
console.log(person); // { name: 'x', surname:
    'y', age: 0, occupation: 'a', vehicle: 'b' }
```

3.2 Access an Object

Accessing hash table object.

- Dot notation person.name
- by indexing person['name']

```
"use strict"

//object person
let person = {
        name : "x",
        surname: "y",
        age: 0,
        occupation: "a",
        vehicle: "b"
}

console.log(person);
/*
   {
        name: 'x',
        surname: 'y',
        age: 0,
        occupation: 'a',
        vehicle: 'b' }
*/

var keys = Object.keys(person)

/*
   [ 'name', 'surname', 'age', 'occupation',
       'vehicle' ]
*/

var myInformation = ['John', 'Doe', 29,
    'Engineer', 'Porsche 911']

let count = 0 //count values

keys.forEach((key) => {
        person[key] = myInformation[count]
        count++
})

console.log(person);

/*
     { name: 'John',
     surname: 'Doe',
     age: 29,
     occupation: 'Engineer',
     vehicle: 'Porsche 911' }
*/
```

3.3 Parsing an object into a function

```
"use strict"

//change card function
var changeCard = (card_) => {
      card_.suit = "Clubs"
}
let card = {
      suit: "Hearts",
      value: "Queen"
}

console.log(card) //{suit: "Hearts", value:
    "Queen"}
changeCard(card)
console.log(card) //{suit: "Clubs", value:
    "Queen"}
```

3.4 Arrays of Objects

```
"use strict"

let cards = [
              {
                    suit : "Hearts",
                    value: "Queen"
              }
]

console.log(cards) //[ { suit: 'Hearts', value:
    'Queen' } ]
console.log(cards[0]) //{ suit: 'Hearts', value:
    'Queen' }
console.log(cards[0].suit) //Hearts
```

accessing array objects.

```
"use strict"

let cards = [

        {
              suit : "Hearts",
              value: "Queen"
        },

        {
              suit: "Clubs",
              value: "King"
        },

        {
              suit: "Diamonds",
              value: "King"
        }

]

let numberOfCards = cards.length //3
for(let i = 0; i < numberOfCards; i++){
        console.log(cards[i].value + " of " +
            cards[i].suit)
}

/*
      Queen of Hearts
      King of Clubs
      King of Diamonds
*/
```

3.5 Built-in Objects

Standard Built-in Objects

Math : random numbers
Date : date objects
String : strings
Number : numbers

3.6 Math Object

Simple game.

```
"use strict"

var guess = (number) =>{
      if (number ==
          (Math.random()*10).toFixed(0)){
              console.log("You chose " + number
                  +" JackPot!!!")
      }else{
              console.log("This round: " +
                  number + ", JackPot number: "
                  +
                  (Math.random()*10).toFixed(0))
           }
}

//Guess number between 0 - 10
let guessNumbers = [1,2,3,4,5,6,7,8,9,10]

guessNumbers.forEach((element) => {
        guess(element)
})

//Lost ten times
/*
This round: 1, JackPot number: 7
This round: 2, JackPot number: 7
This round: 3, JackPot number: 9
This round: 4, JackPot number: 6
This round: 5, JackPot number: 7
This round: 6, JackPot number: 0
This round: 7, JackPot number: 7
This round: 8, JackPot number: 5
This round: 9, JackPot number: 1
This round: 10, JackPot number: 3
*/

//Win
/*
   You chose 9 JackPot!!!
*/
```

3.7 Math truncate

```
"use strict"

//truncate rounds and removes all decimal points
var guess = (number) =>{
        if (number ==
            Math.trunc((Math.random()*10))){
             console.log("You chose " + number
                 +" JackPot!!!")
        }
}

//Guess number between 0 - 10
let guessNumbers = [1,2,3,4,5,6,7,8,9,10]

guessNumbers.forEach((element) => {
     guess(element)
})

/*
   You chose 8 JackPot!!!
*/
```

3.8 Date Object

```
"use strict"

var date = new Date()
console.log(date); //2019-06-15T19:06:25.648Z
```

3.9 toDateString()

```
"use strict"

var date = new Date().toDateString()
console.log(date); //Sat Jun 15 2019
```

4 Programming for web pages

4.1 DOM

Document Object Model: Defines how the data of a web page is organized and manipulated.

Document : HTML file
Model : Data (stored in an object)

4.2 Programming the DOM

```
<!DOCTYPE html>
<html>
   <head>
        <title>Listing1</title>
   </head>
        <body>
        <p id="test">DOM</p>

        <script>
                   let para =
                       document.getElementById('test')
                   console.log(para);
                   para.innerText = "Document
                       Object Model"
        </script>
        </body>
 </html>
```

4.3 Accessing DOM objects using an external file

HTML File : listing2.html

```
<!DOCTYPE html>
<html>
   <head>

        <title>
              Listing2
        </title>

   </head>
              <body>

                   <h1 id="h_1">Listing2</h1>

                   <script
                       src="listing2.js"></script>

              </body>
</html>
```

JS File : listing2.js

```
"use strict"
//class
function Simple(a,b){
   this.a = a,
   this.b = b
}

//method 1
Simple.prototype.sum = function(){
   return (this.a + this.b)
}

//instantiation
var d = new Simple(12,2)
console.log(d.sum())

//manipulating the DOM
var h = document.getElementById('h_1')
h.innerText = "Version " + (d.sum()).toString()
```

4.4 Handling Buttons

HTML file : listing3.html

```
<!DOCTYPE html>
<html>
   <head>
         <title>Listing2</title>
   </head>
   <body>
      <h1 id="h_1">Handling Events</h1>
      <p id="formula">Function used here</p>
      <button type="button" name="ok-button"
           id="ok_button">OK</button>
      <script src="listing3.js"></script>
   </body>
</html>
```

JS file : listing3.js

```
"use strict"

let ok_Button =
    document.getElementById('ok_button')

function M(a,b){
      this.a,
      this.b
}

M.prototype.sum = function(){
      //sum() method from class M(a,b)
      return this.a + this.b
}

ok_Button.addEventListener('click', function(){
      //code here...
      let ff = new M(2,3)
      var f = document.getElementById('formula')
      f.innerText = ff.sum //get function
})
```

4.5 Manipulating DOM object styles

HTML file : listing4.js

```
<!DOCTYPE html>
<html>
   <head>
         <title>Listing2</title>
   </head>
   <body>
      <h1 id="h_1">Handling Events</h1>
      <p id="text">Manipulate this text</p>
         <button type="button" name="ok-button"
              id="clear">
               clear
         </button>
         <button type="button"
              name="revert-button" id="revert">
               Revert
         </button>
      <script src="listing4.js"></script>
   </body>
</html>
```

JS file : listing4.js

```
"use strict"

var clear_button =
    document.getElementById('clear')
var revert_button =
    document.getElementById('revert')
var textToManipulate =
    document.getElementById("text")

function clear(object){
     object.style.display = 'none' //remove
         element
}

function revert(object){
     object.style.display = 'block' //revert
         element
}

clear_button.addEventListener('click',
    function(){
     clear(textToManipulate)
})

revert_button.addEventListener('click',
    function(){
     revert(textToManipulate)
})
```