

# JAVASCRIPT OBJECTS

Thursday, December 6, 2018

7:24 PM

Objects allows a programmer to store related pieces of information at a central location.

```
let myBook = {  
  //property  
  title: "Harry Potter",  
  author: "JK Rowling",  
  numOfPages: "590"  
}  
  
console.log(`${myBook.title}  
was written by  
${myBook.author} and has  
${myBook.numOfPages} pages.`)
```

Here "**myBook**" is an object with **(title, author and numOfPages)** as the properties of the object. One can get the property value using the **dot** notation.

```
let myBook = {  
  //property  
  title: "Harry Potter",  
  author: "JK Rowling",  
  numOfPages: "590"  
}  
  
console.log(`${myBook.title}  
was written by  
${myBook.author} and has  
${myBook.numOfPages} pages.`)  
  
// console.log(myBook)
```

object.

```
myBook.title = 'Fantastic Beasts'

console.log(`${myBook.title}
              was written by
              ${myBook.author} and has
              ${myBook.numOfPages} pages.`)
```

```
thevatsal@Vatsals-MacBook-Pro /PROJECTS/Javascript/all-codes/js-objects node objects-101.js
Harry Potter
    was written by
    JK Rowling and has
    590 pages.
Fantastic Beasts
    was written by
    JK Rowling and has
    590 pages.
```

The above given example shows that we can change the object property also using the **dot** notation.

## OBJECTS AND FUNCTIONS:

Passing an object into a function.

```
let myBook = {
  //property
  title: "Harry Potter",
  author: "JK Rowling",
  numOfPages: "590"
}

let otherBook = {
  //property
  title: "Fantastic Beasts",
  author: "JK Rowling",
  numOfPages: "343"
}

let getSummary = function(book){
  console.log(`${book.title} by ${book.author} has ${book.numOfPages}`)
}

getSummary(myBook)
getSummary(otherBook)
```

notation

Returning an object from a function:

```
let myBook = {
  //property
  title: "Harry Potter",
  author: "JK Rowling",
  numOfPages: "590"
}

let otherBook = {
  //property
  title: "Fantastic Beasts",
  author: "JK Rowling",
  numOfPages: "343"
}

let getSummary = function(book){

  return {
    //object
    summary: `${book.title} by ${book.author}`,
    pageCountSum: `${book.title} has ${book.numOfPages} pages`
  }

  // console.log(`${book.title} by ${book.author} has ${book.numOfPages}`)
}

let bookSummary = getSummary(myBook)
let otherBookSummary = getSummary(otherBook)

console.log(bookSummary.pageCountSum)
console.log([bookSummary.summary])
```

TEMPERATURE CONVERTER USING OBJECT RETURNED INSIDE A FUNCTION

```
let converter = function(allTemp = 32) {
  let C = (allTemp - 32)*(5/9);
  let K = (allTemp - 32)*(5/9)+273.15;

  return {
    tempF: allTemp,
```

```
    tempC: C,  
    tempK: K  
  }  
}  
  
let cel = converter();  
console.log(cel.tempC)  
console.log(cel.tempF)  
console.log(cel.tempK)
```

### OBJECT REFERENCES:

```
let myAccount = {  
  //properties  
  name: "Vatsal",  
  expences: 0,  
  income: 0  
}  
  
let addExpense = function(account, amount) {  
  account.expences = account.expences + amount  
  console.log(account)  
}  
  
addExpense(myAccount, 45)  
console.log(myAccount)
```

```
let myAccount = {  
  //properties
```

```

    name: "Vatsal",
    expences: 0,
    income: 0
  }

  let addExpense = function(account, amount) {
    account = {
      // account.expences = account.expences + amount
      console.log(account)
    }
  }

  addExpense(myAccount, 45)
  console.log(myAccount)

```

Here we will be getting an empty object for first consoleLog and for the second consoleLog the myAccount.

```

let myAccount = {
  //properties
  name: "Vatsal",
  expences: 0,
  income: 0
}

let otherAccount = myAccount

otherAccount.income = 1000

let addExpense = function(account, amount) {
  // account = {

  // }
  account.expences = account.expences + amount
  // console.log(account)
}

```

we will get



```
}  
  
addExpense(myAccount, 45)  
console.log(myAccount)
```

Here we have binded myAccount to otherAccount so when we change one or the other the be reflected in both the acconts.

If we change the whole of any account there won't be any change in the other account, its only change the property of a account it gets reflected in the other account.

```
let myAccount = {
  name: "Vatsal",
  expenses: 0,
  income: 0
}

let addExpenses = function(account, expense) {
  account.expenses = account.expenses + expense
}

let addIncome = function(account, incomeAmt){
  account.income = account.income + incomeAmt
}

let accountReset = function(account){
  account.income = 0
  account.expenses = 0
}

let accountSummary = function(account){
  console.log(`Account for ${account.name} has ${account.income - account.expenses}.
  ${account.income} in income. ${account.expenses} in expenses `)
}

addIncome(myAccount, 8000)
addExpenses(myAccount, 3000)
accountSummary(myAccount)
accountReset(myAccount)
accountSummary(myAccount)
```

## A simple expenses manager.

## METHODS

changes will

only when we

Functions inside an object. Object property whose value is a function. One can get access of properties of the object right inside that function created in the object itself. We will use the **this** keyword to access the properties inside the function/method created in that particular object.

**This** means that, on what object the method is defined on so when we create a method it will be defined on that particular object and thus we will use the **this** keyword to access the properties of that object.

```
let restaurant = {
  name: 'Amb',
  guestCapacity: 75,
  guestCount: 0,
  // method
  checkAvailability: function(partySize){
    let seatsLeft = this.guestCapacity - this.guestCount
    return partySize <= seatsLeft
  }
}

// calling a function inside of a object i.e. a method.
let status = restaurant.checkAvailability(33)
console.log(status)
```

```
let restaurant = {
  name: 'Amb',
  guestCapacity: 75,
  guestCount: 0,
  // method
  checkAvailability: function(partySize){
    let seatsLeft = this.guestCapacity - this.guestCount
    return partySize <= seatsLeft
  },
  seatParty: function(partySize){
    this.guestCount = this.guestCount + partySize
  },
  removeParty: function(partySize){
    this.guestCount = this.guestCount - partySize
  }
}
```

all the  
e **this**  
ect.

will be  
ty values.

```

        this.guestCount = this.guestCount + partySize
    }
}

// calling a function inside of a object i.e. a method.
restaurant.seatParty(70)
restaurant.removeParty(15)
let status = restaurant.checkAvailability(32)
console.log(status)

```

## STRING METHODS

```

let name = 'Vatsal'

// length property

console.log(name.length)

```

```

let name = 'Vatsal Saglani'

// length property

console.log(name.length)

// convert to upper case

console.log(name.toUpperCase())

// convert to lower case

console.log(name.toLowerCase())

```

```
// Includes method -- to check password --
```

```
let password = 'abc123PASSWORD098'  
console.log(password.toLowerCase().includes('password'))
```

```
// Trim method -- remove extra whitespace in the input
```

```
let mt = ' dfdff '
```

```
console.log(mt.length)
```

```
console.log(mt.trim().length)
```

```
log(message?: any, ...optionalParams:
```

## Simple password validator

```
// Password validator
```

```
let isValid = function(password) {  
  if(password.length < 8 || password.toLowerCase().includes('password')){  
    console.log('Invalid password')  
  } else {  
    console.log('Validated!!')  
  }  
}
```

```
isValid('pAssWord1212')
```

```
// Password validator
```

```
let isValid = function(password) {  
  // if(password.length < 8 || password.toLowerCase().includes('password')){  
  //   console.log('Invalid password')  
  // } else {  
  //   console.log('Validated!!')  
  // }  
  return password.length >= 8 && !password.toLowerCase().includes('password')  
}
```

```
console.log(isValid('pAssWord1212'))
```

## NUMBERS

```
let num = 12233.9030

// to fixed method

console.log(num.toFixed(32))

// ROUND METHOD
console.log(Math.round(num))

// Round Down
console.log(Math.floor(num))

// Round Down
console.log(Math.ceil(num))
```

### GENERATING A RANDOM NUMBER:

```
// generates a number between 0 and 1
let min = 10
let max = 20

let randomNum = Math.floor(Math.random() * (max-min + 1)) + min
console.log(randomNum)
```

### GUESSING GAME:

```
let validateGuess = function(num, max = 1, min = 5){
    let random = Math.floor(Math.random() * (max - min + 1)) + min
    return random === num
}
```

```
console.log(validateGuess(2))
```

## DIFFERENT WAYS TO DEFINE VARIABLES IN JAVASCRIPT

**"const":** cannot re-assign a *const* based variable once we created it.

**"let":** can re-assign a *let* based variable once we create anytime.

We can use **const** when we are not going to re-assign variable. Functions can be declared using **const** keyword

While using **const** to declare a object we can re-assign the value of the properties but we cannot re-assign the whole object.

```
// valid

const person = {
  age: 33
}

person.age = 34

console.log(person)

// invalid

person = {}
```



ing the ***const***

cannot re-

## **var** vs **const** vs **let**

- Using **var** we can re-declare a variable that's already been declared.

```
var firstName = 'Vatsal'  
  
// can reassign values  
  
firstName = 'nddk'  
  
var firstName = 'fjjfjf'  
  
console.log(firstName)
```

- **var** is function scoped not block scoped

```
if(10 === 10){  
    var firstName = 'vatsal'  
}  
console.log(firstName)
```

- If we were to use **let** or **const** we would surely get an error because the variable **firstName** is already declared in the global scope or in the scope we are trying to display its value.

```
console.log(age)  
var age
```

***lame*** isn't in

- We won't get any error.
- **var** declaration is hoisted at top of the scope.

```
console.log(age)
var age = 09
```

- Here also we won't get any error but, the output will be **undefined**.

```
age = 10
console.log(age)
var age
```

- Here we will get the log output as 10 no errors.