→ Spaces, Line Breaks & Not important

Js objects

In Js → Almost everything → object.

Primitive value

Immutable (cannot be changed)

: Value has no properties / methods. eg: 3.14

Primitive Data Type

: Data that has

Create Js object

- Single object
  - Object literal
    - keyword – new
  - define object constructor → Create object; → Constructed Type
- Using → Object.create({}) → pg 167

→ **Object literal** : List of `name:value` pairs inside {}

eg:

```
const pee = {
    fi: "John",
    da: "Sagar",
    age: 21.
};
```

150 (pg 92) (pg 158)

new → Create object & then add properties

JUNE 2020 | Week 25 | 167-199 | 15 Monday

eg:
```
const per = new object();
per.fi = "John";
per.la = "Sagal";
per.age = 21;
```

Objects are Mutable → addressed reference not value

eg:
```
const x = person;   // Not create → copy of person
```
object x → Not copy of person → It is person

Both x, person → Same object

eg:
```
const pert = { first: "mj", last: "Desh", age: 51,
               id: "SD001HYU"
             };
const dupobj = pert;
dupobj.age = 64;
```

Object Properties → for in → refer (pg 81) (pg 159)

```
for (let variable in object) {
    //code
}
```
Executed once for each property

```
Const pert = {first: "mj", last: "Desh", age: 51, id: "SD001HYU"};
let txt = "";
for (let key in pert) {
    txt += pert[key] + " ";
}
document.write(txt);
```

Add new properties → `pert.age = 15;`

JUNE						2020
Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

```
let txt = "";
const pert = {first: "Raj", last: "Dean"};
pert.age = 15;
for (let x in pert) {
    txt += pert[x] + " ";
}
console.log(pert);
```

→ **delete** → Keyword
→ deletes property from an object.

eg.

```
const pert = { first: "Raj", last: "Dean" };
delete pert.last;
console.log(pert.last);   //undefined
console.log(pert);
```

→ deletes → both → value → property; After deletion property itself, property can't be used before it's added back again

→ No effect on → variables or Functions } designed to be used on object properties

→ Not use on → Predefined JS object properties. → Will crash the app

→ **Nested objects** → Eg.

```
const pert = {
    first: "Raj",
    last: "Dean"
    cars: {
        car1: "Ford",
        car2: "Bmw",
        car3: "Audi"
    }
};
console.log(pert.cars["car1"]);
console.log(pert.cars);
```

O/p: Ford
{car1: ___ }

→ **Nested Array & object**

eg. j txt
o txt = <h1>...</h1>

Cars

name	models
Ford	- - -
Fiat	- - -

```
let txt = "";
const pert = {first: "Raj", last: "Dean",
    cars: [
        {name: "Ford", models: ["Fiesta", "Focus", "LPa0"]},
        {name: "Fiat", models: ["Fiest", "Fox", "LP991"]},
        {name: "Merc", models: ["Mk1", "P90"]}
    ]
};
for (let i in pert.cars) {
    txt += "<h1>" + pert.cars[i].name + "</h1>";
    for (let j in pert.cars[i].models) {
        txt += pert.cars[i].models[j] + "<br>";
    }
}
document.write(txt);
```

Array of object in object

→ **Object Methods & this Keyword** (pg 90)

→ Accessing Object Methods → Syntax: `objectname.methodname()`

↳ eg.

```
const pert = {
    first: "Raj",
    last: "Dean",
    full: function() {
        return this.first + " " + this.last;
    }
};
alert(pert.full());
```

If → full() → return
↳ full → Function def

JUNE 2020

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

18 Thursday

Add Method to object

```
const pert = {first: "Raj", last: "Dean"};
pert.full = function() {
    return this.first + " " + this.last;
}
console.log(pert.full());
```

can also be

```
return (this.first + " " + this.last).toUpperCase();
```

Using Built in Methods

Display JS objects

o/p → [object object]

Convert object to Array

Object.values()

Eg:

```
const myobj = {
    id: 501,
    pid: "A001",
    name: "Sandra Hellman"
};
const myarr = Object.values(myobj);
console.log(myarr);
```

o/p: (3) [501, 'A001', 'Sandra Hellman']

Comma Sol

Sol

Display name by Object Properties

Loop in Object Properties

JSON.stringify()

Object.values()

Refer this for object literal

JSON.stringify()

```
const myarr = JSON.stringify(myobj);
```

o/p: {"id": "501", "pid": "A001", "name": "..."}

Convert Date into string

JUNE 2020

Week 25 | 171-195

19 Friday

1 Date
- convert date into String

```
const mjobj = {
    name : "Joll",
    today : new Date()
}
const myarr = JSON.stringify(mjobj);
console.log(myarr);
```

2 Functions
- Not Stringify functions

```
const mjobj = {
    name : "Joll",
    age : function(){return 24;}
}
const myarr = JSON.stringify(mjobj);
// const myarr = JSON.stringify(mjobj.age);
undefined
console.log(myarr);
```

o/p : {"name":"Joll"}

Convert Function into Strings before Stringifying ← This is fixed

```
mjobj.age = mjobj.age.toString();
```

o/p "{"name":"Joll","age":"function(){return 24;}"}

3 Array

```
const myoarr = ["John", "Sal", "Tiger"];
let Strinarr = JSON.stringify(myoarr);
console.log(Strinarr);
```

o/p : ["John","Sal","Tiger"]

Object Accessors → EcmaScript5 (ES 5 2009) allow you to define Properties. Computed

e.g. : Next Page

Setter "get Keyword" , Setter → Set Keyword

JUNE 2020

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

20 Saturday | 172-194 | Week 25

Get

```
const objdet = {
    id: 6701,
    name: "JohnCena",
    age: 60,
    address: {
        Street: "St John",
        State: "California",
        country: "USA",
        Pincode: "23-64-7001"
    },
    Profession: "Actor, Wrestler",
    get det(){
        return this.age;
    }
};
document.write(objdet.det);
```

o/p: 60

Set

```
const objdet = {
    id: 6701,
    name: "John Cenath",
    age: "",
    set jage(agedet){
        this.age = agedet;
    }
};
objdet.age = 60;
console.log(objdet.age);
```

o/p: 60

Why use getter setter → Simple syntax → better data quality → useful for doing things behind scenes → allow equal syntax for properties / methods.

(Continued from pg 154)

Object.defineProperty()

→ It can be used to add getters & setters

Eg: Problem →

```
const person = {firstName: 'Mosh', lastName: 'Hamedani'};
console.log(`${person.firstName} ${person.lastName}`);
```

21 Sunday

This has to be repeated → if we want to use it in multiple times.

Sol → functions

```
fullName: function(){}
```

→

```
fullName(){}
```

JUNE 2020

Week 26 | 174-192

22 Monday

```
const person = {
    firstName: "Mosh",
    lastName: "Hamedani",
    fullName() {
        return `${person.firstName} ${person.lastName}`
    }
};
person.fullName = 'John Smith';
console.log(person.fullName());
```

Issue: I don't want to call it as method

Access Properties ← getter ← Get ← Nicer if I treat this as property

change ← Setter ← Set

Eg 1:

fig 7.3 split() → text = How are you ? → How, are, you, ? → split(" ")

H,o,w, ,a,r,e, ,y,o,u, ?

Eg 2:

```
const person = {
    first: "Sam",
    last: "Williams",
    get full() {
        return `${person.first} ${person.last}`
    },
    set full(value) {
        const parts = value.split(' ');
        Console.log(parts); // ['John', 'Moore']
        this.first = parts[0];
        this.last = parts[1];
    }
}
person.full = "John Moore";
Console.log(person);
```

split(" ")

(2) ['John', 'Moore']

{first: 'John', last: 'Moore'}

→ ▷ {first: 'John', last: 'Moore'}

Click on tab you get

first: "John"

full: (...) ← Hover Mouse ← You get

last: "Moore"

▷ get full: f full()

▷ set full: f full(value)

▷ [[Prototype]]: object

Invoke property getter

15.7

23 Tuesday

Object.defineProperty — 175-191 / Week 26

{value : value}

obj → The object on which to define property

Method

Name of property to be defined or modified

* Syntax → Object.defineProperty(obj, prop, descriptor)

def → defines → new property → directly on → object → returns

→ To change flags → use this method

→ Cannot change back → define property → doesn't work

→ Non configurable → raise → Properties defined as being modified ← descriptor for property

→ Return → Return object that passed to function.

Eg:

```
const obj = {counter : 5};
Object.defineProperty(obj, "reset", {
    get : function(){this.counter = 0;}
});
Object.defineProperty(obj, "increment", {
    get: function(){this.counter++;}});
Object.defineProperty(obj, "decrement", {
    get: function(){this.counter--;}});
Object.defineProperty(obj, "add", {
    set: function(value){this.counter += value;}});
Object.defineProperty(obj, "subtract", {
    set: function(value){this.counter = value;}});
console.log(obj.counter); // 5
// Play with counter
obj.reset;
```

158

```
console.log(obj.counter); //0
obj.add = 10;
console.log(obj.counter); //10
obj.subtract = 2;
console.log(obj.counter); //8
obj.increment;
console.log(obj.counter); //9
obj.decrement;
console.log(obj.counter); //8
```

O/P
5
0
10
8
9
8

Final O/P
8

-x-

c) Object Constructors (pg 150, pg 92)

1. In constructor function this not have value — It is substitute for new object
2. Value → this become new object → when new object created
3. Create only single objects; Need Blue print → create many objects of same type → use Object Constructor function
4. Object of same type created by calling constructor function with new keyword

Eg.:

```
function Person(fir, las, age){
    this.fir = fir;
    this.las = las;
    this.age = age;
}
const perobj1 = new Person("Joy", "Raghu", 20);
const perobj2 = new Person("M", "Lopez", 32);

console.log(perobj1.fir);
console.log(perobj2.fir);
```

O/P: Joy
M

25 Thursday — To object — 177-189 | Week 26

1 — Add Property (refer pg 160) — Add new property to existing object is easy.

```
function Person(fir){
  this.fir=fir;
}
const perobj1 = new Person("Jay");
const perobj2 = new Person("Ali");
perobj1.nation = "India";
perobj2.nation = "Srilanka";
console.log(perobj1.nation);
console.log(perobj2.nation);
```

o/p: India
Srilanka

2 — Add Method — Add new method to existing object is easy

(pg 165) — o/p: Mona Ray

```
function Person(fir, last){
  this.fir = fir;
  this.las = las;
}
const perobj1 = new Person("Mona", "Ray");
const perobj2 = new Person("Vira", "Sagar");
perobj1.full = function(){
  return this.fir + " " + this.las};
console.log(perobj1.full());
```

3 — Add Property to Constructor — You cannot add new property to object constructor the same way you add new property to existing object

JUNE 2020 — Week 26 | 178-188 — 26 Friday

o/p: undefined

To add, you must add it to constructor function

```
function Person(fir){
  this.fir = fir;
}
const perobj1 = new Person("Mona");
const perobj2 = new Person("Vira");
Person.element = "fire";
console.log(perobj1.element);
```

So,

```
function Person(fir, element){
  this.fir = fir;
  this.element = element;
}
const perobj1 = new Person("Mona", "Fire");
const perobj2 = new Person("Vira", "Air");
console.log(perobj1.element);
```

o/p: Fire

4 — Add Method to Constructor — Can also define methods in constructor function

```
function Person(fir, zodiac){
  this.fir = fir;
  this.zodiac = zodiac;
  this.zodiacchange = function(name){
    this.zodiac = name;
  }
}
const perobj1 = new Person("Mona", "Pisces");
perobj1.zodiacchange("Gemini");
console.log(perobj1.zodiac);
```

o/p: Gemini

JUNE 2020

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

15 16

27

179-187 | Week 26

Saturday

Built In Js Constructors

Math()

Global object

So new keyword on it

Strings() → "a"

Objects() → {}

Numbers() → 50

Booleans() → true/false

Arrays() → []

RegExp() → /()/

Functions() → () {}

Dates()

Strings

RW

Pg 112

Target

Pill

Click

Click me

Black

Bubbling

28 Sunday

Capturing

162 Object Prototypes Also pg 175

All → Is object → inherit → Properties, Methods → from

Date object ← Date Prototype
Array object ← Array Prototype
Person object ← Person Prototype

Object.Prototype ← Top of ← Prototype Inheritance Chain

Inherit from ← Date, Array etc object

Prototype property ← Allow → to add ← new properties, methods

Object Construction ↑ to

```
function Person(fir, las, age){
    this.fir = fir;
    this.las = las;
    this.age = age;
}
const Peru = new Person("Manoj", "Javed", 21);
Person.fayplace = "goa";
console.log(Peru.fayplace); // undefined
```

o/p: goa

Person.prototype.fayplace = "goa"

Modify this line to

```
function Person(fir, las, age){
    this.fir = fir;
    this.las = las;
    this.age = age;
}
const Peru = new Person("Manoj", "Javed", 21);
Person.prototype.name = function(){return this.fir + " " + this.las;};
console.log(Peru.name());
```

o/p: Manoj Javed

o/p: Uncaught: Type Error

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

30 Tuesday

→ Only modify your own properties

→ Never modify standard JS objects of prototypes.

→ **Object Iterables**: Objects iterated over with (for of) → implement (technically) Symbol.iterator()

for of → pg 81

→ **Iterate over String**:

```
const name = "W3schools";
let txt = "";
for (let x of name) {
    txt += x + "<br>";
}
document.write(txt);
```

→ **Array**:

```
const name = [1,2,3,4,5];
let txt = "";
for (let x of name) {
    txt += x + "<br>";
}
document.write(txt);
```

→ **JS Iterables / Iterators**:

→ **Iterator Protocol** → define how to produce sequence of values from object.

→ **next()**: Return object with 2 properties → value (Next value), done (true / false)

→ **Object become Iterator**: When it implements next()

→ **Properties (next())**:
- **value** → returned by iterator; can omit if done = true
- **done** → true if iterator completed; false if iterator produced new value.

01 Wednesday

n = 0; return {} ; next() → {5, false} (n = 0 + 5)

→ Return with {}

→ **Home Made Iterable** (not support for of)

Eg: Iterable → Never ending (10, 20, 30...) Everytime next() called

```
function mynum() {
    let n = 0;
    return {
        next: function() {
            n += 5;
            return { value: n, done: false };
        }
    };
}
const itemy = mynum();
itemy.next();    // 5
itemy.next();    // 10
console.log(itemy.next().value);  // 15
```

o/p: 15

→ **JS Iterable**: Is an object that has Symbol.iterator → is function → returns next() → auto called by (for of)

Eg:

```
let text = "";
const myuns = {};
myuns[Symbol.iterator] = function() {
    let n = 0;
    done = false;
    return {
        next() {
            n += 10;
            if (n == 60) {done = true}
            return {value: n, done: done}
        }
    };
}
for (const xg of myuns) { text += xg + "<br>"; }
document.write(text);
```

o/p: 10, 20, 30, 40, 50

JULY 2020

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

for eg:

```
let numbers = [1,2,3];
console.log(numbers);
```

02

184-182 | Week 27

Thursday

JUL 202

Running function

```
let numbers = [1,2,3];
console.log(numbers[Symbol.iterator]);
```

ƒ values() { [native code] }

Array iterator {}

```
Console.log(numbers[Symbol.iterator]());
```

consist of property which is next() function

```
let numbers = [1,2,3];
let itfn = numbers[Symbol.iterator]();
console.log(itfn.next());
console.log(itfn.next());
console.log(itfn.next());
console.log(itfn.next());
```

O/P

{value: 1, done: false}

{value: 2, done: false}

{value: 3, done: false}

{value: undefined, done: true}

```
let numbers = [1,2,3];
numbers[Symbol.iterator] = function() {
  return {
    next: function() {
      return {
        done: false,
        value: 10
      };
    }
  };
};
let itfn = numbers[Symbol.iterator]();
console.log(itfn.next());
console.log(itfn.next());
```

O/P

{done: false, value: 10}

{done: false, value: 10}

Next Page

Variable = (next > 15) ? 'true' : 'false'

cond

Ternary operator

```
let numbers = [1,2,3];   //or let numbers = [];
numbers[Symbol.iterator] = function() {
    let nextValue = 10;
    return {
        next: function() {
            nextValue ++;
            return {
                done: nextValue > 15 ? true : false,
                value: nextvalue
            }
        }
    }
}
for (let value of numbers) {
    console.log(value);
}
```

o/p
11
12
13
14
15

~) Object Methods
pg (167)

Object create() pg 149
- keys() / values() → pg 153
- entries() ⊗
- assign() ⊗
- freeze() ⊗
- seal() ⊗

obj [prop name]

~) Interview (I)

1) Pallindrome Algorithm

Get (String / Number) from ~ d
↓
Take a Temporary variable → holds
↓
Reverse a given → d
↓
Compare (Original, Reversed)
↓
If same → No or String is Pallindrome

```
function pallindromeCheck (str) {
    const arrval = str.split("");
    const revval = arrval.reverse();
    const revstr = revval.join("");
    if (String == revstr) { // Success } else { // Error }
}
const String = prompt("enter");
const val = ...
```

JULY					2020	
Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Object Methods

(Pg no 149, Pg no 166)

04 Saturday — JULY 2020 — 186-180 | Week 27

S.No.	Name	Details
1	Object.create()	Create object with link to existing object (prototype).
2	Object.keys()	Create Array containing keys of an object.
3	Object.values() (Pg 163)	Create Array containing value of an object.
4	Object.entries()	Create Nested array of key/value pairs in object.

Syntax: `Object.create(prototype, [, propertiesObject])`

```
Const person = {
  isHuman: false,
  printIntroduction: function() {
    console.log(
      `My name is ${this.name}.
       Am I human? ${this.isHuman}`
    );
  }
};

const me = Object.create(person);
me.name = "Manu";
me.isHuman = true;
me.printIntroduction();
person.printIntroduction();
```

My name is Manu. Am I human? true

05 Sunday

JULY 2020 — Week 28 | 188-178 — 06 Monday

```
Const person {
  name: "Sonal",
  sign: "Leo",
  element: "Water",
  number: 8
};
console.log(Object.entries(person));
```

```
[["name", "Sonal"], ["sign", "Leo"], ["element", "Water"], ["number", 8]]
```

5 Object.assign()

Used to copy value from one object to another.

Syntax: `Object.assign(target, source)` (Obj1, Obj2) — from, to

```
Const zodiac = {
  element: "Water",
  sign: "Leo"
};
const name = {
  number: 8,
  name: "Sonal"
};
const char = Object.assign(zodiac, name);
console.log(char);
const dark = Object.assign(name, zodiac);
console.log(dark);
```

```
{element: 'Water', sign: 'Leo', number: 8, name: 'Sonal'}
{number: 8, name: 'Sonal', element: 'Water', sign: 'Leo'}
```

6 Object.freeze()

Prevents modifications to properties and values of an object.

- Properties from being added.
- Properties from being removed from object.

Syntax: `Object.freeze(obj)`

JULY 2020

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

9 It returns passed object

10 It does not create frozen copy

Eg 1) //Create Object

```
const obj1 = {data: 'Processing'};
//Create second object which freeze properties of 1st object
const obj2 = Object.freeze(obj1);
// updating properties of frozen object
obj2.data = 'Pending';
// Display property of frozen object
console.log(obj2.data);
```

Processing O/P

Eg 2)

```
const user = {name: "metatag", password: "1998"};
const newUser = Object.freeze(user);
newUser.password = "test";
newUser.active = true;
console.log(newUser);
```

{username: 'metatag', password: '1998'} O/P

7) Object.seal() => Prevents new properties from being added to objects

But => Allow Modification of existing properties

1/10

JULY
2020

Week 28 | 190-176

08
Wednesday

) Change in (object freeze)
| Object.freeze(user); |
| Object.seal(user); |

You get o/p: {username: "mdrunning",
password: "*******"
}

↑ Modified Property (value)

M) Event Delegation: Manage events ...