Java Script

Written By: Ibrahim Abdelaty

الكـاتب : ابراهيـم عبــد العــاطـي

حسابي علي تويتر: https://twitter.com/IbrahimElaty

Intro

- يكتب ملف Java Script داخل العنصر <script> </script> **او** <script src= "main.js" > </script>
- يستحسن وضع العنصر <script src= "main.js" > </script> اسفل العنصر الذي يتطبق عليه الكود الموجود داخل الملف ... او في نهاية عنصر الـ body.
- نضع (;) في نهاية كل سطر في الجافا اسكريبت.

Java Script Data types

اذا اردت اظهار نوع البيانات (في الـ console) نكتب ... console.log(**typeof** "Ibrahim");

- "Ibrahim" : **String**
- 4410 : **number**
- ["ib" , "rah" , "im"] or [12 , 5 , 33 , 20] : **array**
- {name: "Ibrahim" , age: 21 , country: "eg" } : **object**
- true or false : **Boolean**
- **Undefined**
- **Null**

Variables

- العناصر المعرفة بـ id في الـ html .. تكون قيمة الـ id تكون اسم متغير يمكن استدعاءه والعمل عليه في الـ JS ..

```
<p id="hell" > Hello <p>
Console.log(hello)     =>  <p id="hell" > Hello <p>
```

- يجب الاعلان عن المتغير قبل استخدامه.
- تسمية الـ identifiers (اسماء المتغيرات) تكتب بالـ camel case.
- تستخدم \ لتخطي حرف مثل " ..... و **\n** لسطر جديد... و + لـ (concatenation) ..

- **Var** => يمكن الاستغناء عن الـ **var** في تعيين قيمة لمتغير..
  - Redeclare (yes) :
  - Variable scope drama (added to window object)

- **let**
  - Redeclare (no) :
  - Variable scope drama (don`t added to window object)

- **const**
  - Redeclare (no) :
  - Variable scope drama (don`t added to window object)

```
var user= "ibrahim";

var user= "ibrahim", age= 21 ;
```

Template Literals

هي طريقة جديدة لـ concatenation في الـ ES6 ... (لجمع السلاسل النصية .. والمتغيرات.. وتخطي الحروف الخاصة عدا السطر الجديد) ... ` .... `

```
x = "Ibrahim";

y = 21;

Console.log( ` my "name" is: ${x}

,and my age is: ${y}` )
```

```
my "name" is: Ibrahim

,and my age is: 21
```

لاحظ: تستخدم **${...}** لاضافة المتغيرات او اضافة دالة (function) (او متغير مطبق عليه دالة)...

Nullish coalescing operator

- ` ${ X || 100 } ` => تكتب قيمة 100 اذا كانت قيمة الـ X .. Null او undefined او 0 او "" او ..
- ` ${ X ?? 100 } ` => تكتب قيمة 100 اذا كانت قيمة الـ X .. Null او undefined.

String Methods

let **x** = " Ibrahim "

- **x.length** 9 => ترجع طول السلسلة...
- **x.repeat(**2**)** Ibrahim Ibrahim => لتكرار السلسلة ...
- **x.trim()** "Ibrahim" => لازالة المسافات من بداية ونهاية السلسلة...
- **x.toUpperCase()** " IBRAHIM " => لتحويل السلسلة النصية الي احرف كبيرة...
- **x.tolowerCase()** " Ibrahim " => لتحويل السلسلة النصية الي احرف صغيرة...
- **x[**2**]** b => للوصول الي حرف عن طريق الفهرس (يبدأ من الصفر)
- **x.charAt(**2**)** b => للوصول الي حرف عن طريق الفهرس (يبدأ من الصفر)
- **x.indexOf(** "r" **,** 1**)** 3 => للوصول الي فهرس حرف (او كلمة) ما ...

  x.indexOf("r" **,** 8**)** -1 => للوصول الي فهرس حرف (او كلمة) ما ... (1- اذا كانت القيمة غير موجودة)
- **x.lastIndexOf(** "i" **)** 6 => للوصول الي فهرس حرف (او كلمة) ما (يبدأ من الخلف) ...
- **x.slice(**2**,** 5 **)** bra => للوصول الي جزء من الكلمة ...

  x.slice(-5 **)** ahim => للوصول الي جزء من الكلمة ...
- **x.split(**"a"**)** ["ibr" , "him"] => للتقسيم من عند عرف معين ...

  x.split("a" , 1 **)** ["ibr"] => لتحديد (limit) للقص ...

let **y** = "Ibrahim Abdelaty"

- **y.subString(**2 , 6**)** rahi => للوصول الي جزء من السلسلة ...

  y.subString(6 , 2**)** rahi => اذا كانت البداية اكبر من النهاية يتم تبديلهما ... واذا كانت البداية سالبة تعتبر صفر
- **y.subStr(**2 , 7**)** rahim A => للوصول الي جزء من السلسلة .. (البداية و عدد الحروف المراد حسابه) ..

  y.subStr(-3 , -1**)** at => تقبل قيم سالبة ...
- **y.includes(** value **,** start [opt]**)** true or false => هل تحتوي السلسلة علي حرف (كلمة) ما ..

- y.startsWith(value , start [opt]) => هل تبدأ السلسلة بحرف (كلمة) ما .. true or false
- y.endsWith(value , length[opt]) => هل تنتهي السلسلة بحرف (كلمة) ما .. true or false

لاحظ: - 10 == "10" => true **اما** 10 === "10" => false

في الاولي يتم مقارنة القيمة فقط .... وفي الثانية يتم مقارنة القيمة والنوع

- Not (!) و or (||) و And (&&)

Arithmetic Operators

الجمع + ، الطرح - ، الضرب * ، القسمة / ، باقي القسمة % ، الأس **

- increment: اذا كانت X = 2 فعند كتابة console.log(++X) تزداد قيمة المتغير 1 ثم تطبع.
  اما console.log(X++) تطبع قيمة المتغير اولا ثم تزداد قيمة المتغير.
- Decrement: اذا كانت X = 2 فعند كتابة console.log(--X) تقل قيمة المتغير 1 ثم تطبع.

لاحظ:

- console.log(+"100") <= 100
- console.log(+"ibrahim") <= NaN (not a number)
- console.log(+true) <= 1
- console.log(- true) <= -1

هناك function خاصة بتحويل البيانات الي اعداد وهي **Number()**

console.log(Number("100")) <= 100

لاحظ:

- يمكن كتابة الارقام بأكثر من طريقة:

1000000 ... 1_000_000 ... 10 ** 6 ... 1e6

Number Methods

- (100).toString() => لتحويل البيانات الي سلسلة نصية يمكن كتابتها هكذا 100..toString() "100"
- 100.555.toFixed(2) => لتقريب الارقام العشرية لعدد خانات معينة (الناتج سلسلة نصية) "100.56"
- parseInt("100.6") => لتحويل البيانات الي اعداد صحيحة .. 100

- **parseFloat(**"100.5"**)** 100.5 => لتحويل البيانات الي اعداد صحيحة او عشرية ان كانت عشرية..
- **Number.isInteger(**15**)** true => هل البيانات اعداد صحيحة ام لا..
- **Number.isNaN(** "Ibrahim"/15 **)** true => هل البيانات (not a number) ام لا..

Math Object

- **Math.round(** 100.65 **)** 101 => لتقريب الاعداد لاقرب رقم صحيح (الناتج رقم)
- **Math.ceil(** 100.3 **)** 101 => يضاف واحد اذا وجد اي كسر عشري (الناتج رقم)
- **Math.floor(** 100.99 **)** 100 => لا يضاف واحد اذا وجد اي كسر عشري (الناتج رقم)
- **Math.max(** 10 , 50 , -99 **)** 50 => اكبر رقم في مجموعة ارقام (الناتج رقم)
- **Math.min(** 10 , 50 , -99 **)** -99 => اكبر رقم في مجموعة ارقام (الناتج رقم)
- **Math.pow(** 2 , 3 **)** 8 => رقم مرفوع لأس ما (الناتج رقم)
- **Math.random()** 5.658 => رقم عشوائي (الناتج رقم)
  Math.random() * 3 لتحديد حد اقصي صحيح (3) ...
- **Math.trunc(** 100.65 **)** 100 => اخذ الجزء الصحيح فقط من العدد (الناتج رقم)

Array Methods (1)

myArray = **[** 'Ibrahim' **,** 'Mohamed' **,** 'Ahmed' **]**

- **Array.isArray(**myArray**)** true => (هل Array ام لا)...
- myArray**.length** 3 => ترجع عدد عناصر الـ Array...
- myArray**.unshift(** "Wael" **,** .. **)** => لاضافة عنصر (عناصر) الي بداية الـ Array...
- myArray**.push(** "Wael" **,** .. **)** => لاضافة عنصر (عناصر) الي نهاية الـ Array...
- myArray**.shift()** => لازالة اول عنصر في الـ Array (ويرجعه) ...
- myArray**.pop()** => لازالة اخر عنصر في الـ Array (ويرجعه) ...

- myArray.**sort()** => لترتيب عناصر الـ Array (حسب محتوي الخانة الاولى) ...
- myArray.**reverse()** => لعكس ترتيب عناصر الـ Array ...
- myArray.**indexOf(** search element , from index [opt] **)** => ترجع فهرس عنصر(1) مبحوث
- myArray.**lastIndexOf(** search element , from index [opt] **)** => يبحث من الاخر ..
- myArray.**includes(** search element , from index [opt] **)** => اذا كان العنصر موجود .. true
- myArray.**slice(** start [opt] , end [opt] **)** => للوصول الي جزء من الـ Array (تنشأ Array جديدة)
- myArray.**splice(** start , delete Count [opt] , items to add [opt] **)**
  => لحذف او اضافة جزء من الـ Array (تغير في الـ Array)
- myArray.**concat(** ArrayOne , ArrayTwo , ... , elementOne , ... **)**
  => لدمج عدد من الـ Arrays والعناصر ...
- myArray.**join(**Separator **)** => لضم عناصر الـ Array معا وتحديد ما هو الفاصل بينهم ...

Object

يحتوي الكائن علي دوال (methods) و خواص (properties) .. يمكن وضع كائن داخل اخر ...

```
let user = {
        // properties
      myName : "Ibrahim",
      myAge : 21,
       //Methods
      sayHello : function () {
      return `Hello` ;
      } , };
console.log(user.myName)     => Ibrahim
console.log(user.sayHello() )  =>  Hello
```

لاحظ:

1) اذا كان اسم الخاصية مكون من كلمتين مثل "my Name" فلا نستطيع الوصول اليها بالـ Dot notation هكذا "user.**"my Name** .. وانما نستخدم الـ Bracket Notation هكذا .. user["my Name"]

2) يمكن وضع كائن داخل كائن (nested objects).. let objectOne = { objectTwo : { ... } }

3) لاستدعاء قيمة خاصية (حتي داخل الكائن) ..... objectName.myName

4) لاضافة او تعديل خاصية للكائن (من خارج الكائن)... objectName.myName = "mohamed";

5) لاضافة دالة للكائن ... **user**.myFunction = **function** ( ){ Block of code }

6) يمكن انشاء كائن جديد بهذه الطريقة ... let myObject = new Object(Any Object or { .. }) ;

7) للوصول الي الـ Object Keys (في Array)... **Object.keys(**myObject**)**;

this keyword

1. تعود علي (تعبر عن) الكائن window .. في السياق العام.

console.log(this) => window{ .. }

2. تعود علي (تعبر عن) مالك الدالة (العنصر الذي تطبق عليه الدالة)... في حالة تطبيق دالة علي عنصر html.

3. تعود علي (تعبر عن) مالك الدالة (الكائن الذي يحوي الدالة) ... في حال كانت الدالة داخل كائن ما.

Object.create( )

تستخدم لانشاء كائن .. مع امكانية استخدام كائن اخر ك (نموذج) في انشاء هذا الكائن الجديد..

let myObject1 = **Object.create(**prototype object **)** ; => create prototype object

let myObject2 = **Object.create(**myObject1 **)** ; => create object from prototype object

لاحظ:

1) يرث الكائن الجديد كل خواص ودوال (الكائن النموذج) بنفس قيمها ... ما لم يتم تغيرها..

Object.assign( )

تستخدم لانشاء كائن من محتوي كائن (كائنات) اخرى.. او اضافة محتوي كائن (كائنات) اخرى الي كائن موجود.

let myObject = **Object.assign(** targetObject or {} , obj1 , obj2 , ... **)** ;

لاحظ :

1) في حالة وجود خاصية او دالة في كل من obj2 , obj1 .. يأخذ القيمة من الاول.

2) في حالة وجود خاصية او دالة في targetObject و اى كائن اخر.. يأخذ القيمة من الكائن الاخر.

Set Data Type

- Can store any data values
- Have size property & Can use forEach
- Have keys, values and entries

لإنشاء (مجموعة) من البيانات ... (**لا يمكن الوصول لعناصر المجموعة بإستخدام الفهرس**) ..

```
let myset =new Set( [1 , 1 , 2 , "a" , "a"] ) ;

console.log(myset );       => { 1 , 2 , a }
```

- myset.**size** لمعرفة عدد عناصر المجموعة ...
- myset.**add(** "b" **)** لاضافة عنصر للمجموعة ...
- myset.**delete(** "b" **)** لحذف عنصر من المجموعة ... يرجع true / false لتبين ان العنصر موجود ام لا.
- myset.**clear( )** لحذف كل عناصر المجموعة ...
- myset.**has(** "b" **)** لمعرفة هل العنصر موجود في المجموعة ام لا ... يرجع true / false

لاحظ: لعمل iteration علي المجموعة ....

```
let iterator = mySet.keys();

console.log( iterator.next() );   => { value: 1 , done: false }

console.log(iterator.next().value );     => 1

console.log(iterator.next().value );     => 2
```

Weak Set

- Collection of objects only
- Store objects and removes them once they become inaccessible
- Dose not have size property & Cannot use forEach
- Dose not have clear, keys, values and entries

```
let myWS =new WeakSet([ { A: 1 , B: 2 } or myObject ] ) ;
```

Map Data Type

- Does not contain keys by default
- Key can be anything [function , object , any primitive data type]
- Ordered by insertion
- Get items by size
- Can be directly iterated
- Better performance when add or remove data

```
let myMap =new Map( [ "name" , "Ibrahim" ] , [ "age" , 21 ] , … ) ;
```

- myMap.**size** لمعرفة عدد عناصر الـ Map ...
- myMap.**set(** key **,** value **)** لاضافة عنصر الي الـ Map ...
- myMap.**get(** key **)** لحصول علي قيمة عن طريق المفتاح (key) الخاص بها ...
- myMap.**delete(** "name" **)** لحذف عنصر من الـ Map عن طريق الـ key... يرجع true / false .
- myMap.**clear( )** لحذف كل عناصر الـ Map.
- myMap.**has(** "age" **)** لمعرفة هل العنصر موجود في الـ Map ام لا ..عن طريق الـ key ...

: Object

- has default keys
- Keys can be string or symbol

لاحظ: لانشاء كائن فارغ ... لا يحتوي علي (properties) / default keys ... null prototype

```
let emptyObj = Object.create(null )
```

Weak Map

- Key can be object only
- Key deleted when the reference delete

If Conditions (control flow)

```
if ( condition ) { block of code }
else if ( condition ) { block of code }
else ( condition ) { block of code }
```

الحالة المختصرة:

```
Condition  ?  if true : if false ;
```

Switch Statement

```
switch ( expression ) {
    case 1:
        block of code
        break;
    case 2:
        block of code
        break;
    Default :
        block of code
        break; }
```

تستخدم افضل من if اذا كانت هناك حالات كثير (مثل ايام الشهر)..

حيث يتم التعبير عن متغير .. والكود الذي يطبق هو الموجود في الحالة التي تساوي قيمة المتغير.. واذا لم توجد يطبق الكود الموجود في القيمة الافتراضية..

لاحظ: يمكن تعيين كود واحد لحالتين ...

```
case 5:
case 6:
    block of code
    break;
```

Loop

for

```
for ( [1] ; [2] ; [3] ){ Block of code }

for ( let i = 1 ; i <= 10 ; i++ ){
    console.log( i )
}                    => 1  2  ...  10
```

while

```
while ( condition ){ Block of code }
```

يستمر الـ Loop طلاما كان الشرط true ...

Do while

```
do {
    Block of code
} while ( condition );
```

يتم تنفيذ الكود الموجود في do بغض النظر عن قيمة الشرط الموجود في while....

Loop control

- break :

يمكن ايقاف التكرار عند تحقق شرط معين عن طريق وضع الشرط داخل الـ Loop ...

```
for ( [1] ; [2] ; [3] ){
    Block of code ;
    if ( condition ){
        break;
    } }
```

- continue :

لتخطي قيمة (قيم) من التكرار عند تحقق شرط معين عن طريق وضع الشرط داخل الـ Loop ..

- Label :

يمكن تعيين Label لل Loop باي اسم تختاره (واستخدامه في التحكم في الـ Loop) ...

```
mainLoop: for ( [1] ; [2] ; [3] ){
    Block of code ;
    if ( condition ){
        break mainLoop ;
    } }
```

Function

```
function myFunc ( parameter , .. ){ Block of code};
```

استدعاء الدالة `myFunc( Argument , .. )`

لاحظ:

1. تستخدم return لاعادة قيمة ما .... تكون اخر شيئ في الدالة ولا يعمل اي شيئ بعدها...
2. القيمة الافتراضية لاي parameter هي undefined ويمكن تغيير القيمة الافتراضية كتالي:

```
.... ( parameter = "value" , .. ) { ...
```

3. في حالة اردت وضع عدد غيرمحدد من المعاملات نضع ثلاث نقاط امام اسم الـ parameter (عبارة عن Array)..

```
.... ( ...param ) { ...        => Rest parameter
```

4. يمكن استخدام الـ parameters مع Rest parameter ..... بحيث تكون Rest parameter في الاخر.
5. يمكن كتابة دالة داخل اخرى (nested functions) ...
6. اي متغير يتم انشاءه داخل دالة سواء بستخدام var او let ... لا يمكن الوصول اليه الا داخل الدالة.
7. في الـ [if, switch , for , while , ..] Block Scope: اى متغير يتم انشاءه بستخدام var .. يكون global ، اما اى متغير يتم انشائه بـ (const) let .. يكون local.

Anonymous Function

هي دالة لا يكون لها اسم (يمكن تخزينها في متغير واستدعائها باسمه ... او تقوم بشئ لمرة واحدة)

```
let myFunction = function ( parameter , .. ){ Block of code};
```

Arrow Function

هي دالة لا يكون لها اسم (يمكن تخزينها في متغير واستدعائها باسمه ... او تقوم بشئ لمرة واحدة)

```
let myFunction = ( parameter , .. ) => { Block of code};
```

لاحظ: اذا كان هناك معامل واحد فقط يمكن ازالة () .. واذا كان هناك سطر واحد يمكن ازالة {} .. و الـ return .

Higher Order Functions

- myArray.**map(** myFunction**)**

تستخدم لتمرير قيم مصفوفة (Array) .. الي دالة .. والناتج يكون Array جديدة.

```
let myVar = myArray.map( function ( element , index [opt], array[opt] ) { ... } , this[opt] )
```

=> element : العنصر الحالي..

=> index : فهرس العنصر الحالي..

=> array : المصفوفة (myArray) ...

=> this: لتعيين قيمة للـ this يمكن استخدامها في الدالة ....

```
let  myArray = [ 1 , 2 , 3 , 4 , 5 ]

let myVar = myArray.map( function (ele) {

        return ele + ele ;

 } ) ;

 console.log(myVar)        => [2, 4, 6, 8, 10]
```

لاحظ: يمكن استخدام اي نوع من انواع الدوال داخل الـ map مثل (.. .. Arrow Function)...

- myArray.**filter(** myFunction**)**

الفرق بينها وبين الـ map .. انها تحتوي علي شرط اذا تحقق تخرج العنصر .. واذا لم يتحقق لا تخرجه.

```
let  myArray = [ 1 , 2 , 3 , 4 , 5 , 6]
let myVar = myArray.filter ( function (el) {
    return el % 2 === 0 ;
} ) ;
   console.log(myVar)        => [2, 4, 6 ]
```

- myArray.**reduce(** myFunction**)**

تستخدم لتمرير قيم مصفوفة (Array) .. الي دالة .. والناتج يكون Array جديدة.

```
let myVar = myArray.reduce( function ( Accumulator , current value , index[opt] , array[opt] ) { ... } ,
initialValue )
```

=> Accumulator: القيمة المخزنة (تكون قيمتها في البداية هي .. initialValue)..

=> current value : العنصر الحالي..

=> index : فهرس العنصر الحالي (يبدأ من 1 اذا كانت Accumulator قيمتها او عنصر اي لم تحدد) ..

=> array : المصفوفة (myArray) ...

=> initialValue: تكون اول عنصر في المصفوفة اذا لم تحدد لها قيمة..

```
let  myArray = [ 1 , 2 , 3 , 4 , 5 ]
let myVar = myArray.reduce( function ( acc , cur ) {
        return acc + cur ;
} );
console.log(myVar)                => 15
```

- myArray.**forEach(** myFunction**)**

لا تنشأ Array جديدة .. ولا يمكن استخدام return معها .. تستخدم مثلا في (اخفاء واظهار عناصر الصفحة).

```
let myVar = myArray.forEach(function ( element , index [opt], array[opt] ) { ... } , this[opt])
```

Array methods (2)

- لإنشاء array من Iterable ... `Array.from( Iterable , MapFunc [Opt] , this [Opt] )`

```
Array.from( “Ibrahim” )          => [ ‘I’, ‘b’, ‘r’, ‘a’, ‘h’, ‘I’, ‘m’ ]
Array.from( { 1, 2, 3, 4, 5, 6, 7 } )     => [ 1, 2, 3, 4, 5, 6, 7 ]
```

لاحظ: يمكن استخدام دالة (function) لتطبيقها علي عناصر المصفوفة قبل إنشائها ...

```
Array.from( “12345” , (n) => +n + +n )      => [ 2, 4, 6, 8, 10 ]
```

- لنسخ جزء من الـ Array الي مكان اخر داخلها (تحافظ علي عدد العناصر)... (تغيير في الـ Array الاصلية)

```
Array.copyWithin( Target index , Start index [Opt] , End index [Opt] );
```

```
const myArray = [ 1 , 1 , 2 , 3 , 4,'a','b' ];
myArray.copyWithin( 2 , -2 , -1 );  =>[1, 1, 'a', 3, 4,'a' ,'b']
```

- لتطبيق دالة (شرطية) علي عناصر الـ Array وترجع True / False .. (هل تحقق الشرط علي احد العناصر ام لا)

```
myArray.some( Function , this );
```

```
let myArray  = [1,2,3,4,5,6] ;
let myNumber =  3;
let check = myArray.some(function (e) {
    return  e > this ;
} , myNumber);
console.log(check);     => true
```

- لتطبيق دالة (شرطية) علي عناصر الـ Array وترجع True/ False.. (هل تحقق الشرط علي كل العناصر ام لا)

```
myArray.every( Function , this );
```

لاحظ: يمكن ارجاع الـ Arguments التي تمرر للدالة عن طريق ...

```
function testArgs () {
return arguments };
```

Destructuring

: Array (1

```
let myFriends = [“Ibrahim” , “Mohamed” , “Wael” , “Mahmoud” ];

let [ a , , c ] = myFriends ;

console.log( a );      => Ibrahim

console.log( c );      => Wael
```

لاحظ: يمكن تعيين قيمة افتراضية يتم الرجوع لها اذا لم يكن مكان المتغير موجود في الـ Array ...

```
[ a , b , c , d , e =”Ahmed” ] = myFriends ;

console.log( e );      => Ahmed
```

لاحظ: لتبديل قيمتى متغيرين (Swapping Variables) : ...

```
let name =13 ;

let age =”Ahmed” ;

[ name , age ] = [age , name ] ;

console.log( name );      => Ahmed
```

: Object (2

```
let user = {
    theName : “Ibrahim”,
    theAge : 21,
    theTitle : “Developer”,
    theCountry : “Egypt”,
    skills: {
        html: 90,
        css: 80, },};

const { theName: n , theAge: a , theCountry: c , skills { html: h }, } = user;

console.log( n );              => Ibrahim
```

```
console.log( c );                => Egypt
console.log( h );                => 90
```

لاحظ: يمكن تعيين قيمة افتراضية يتم الرجوع لها اذا لم يكن مكان المتغير (property) موجود في الكائن ...

```
const { theName , theAge , theCountry: c  , theColor: o = "red", } = user;
console.log( o );                => red
```

لاحظ: يمكن عمل Destructuring بهذه الطريقة ...

```
( { theName , theAge, theCountry, skills { html}, } = user );
console.log( theName );        => Ibrahim
```

Document Object Model (DOM)

عند تحميل الصفحة .. ينشأ المتصفح نموذج للصفحة عبارة عن كائن (يحتوي علي جميع العناصر).. وباستخدام الـ js يمكنك التحكم الكامل في هذه العناصر..

:DOM Selectors

- `document.getElementById("logo");` للوصول لعنصر باستخدام الـ id .
- `document.getElementsByClassName("container");` للوصول لعنصر (عناصر) باستخدام الـ class
- `document.getElementsByTagName("div");` للوصول لـ (عناصر) باستخدام اسم العنصر.
- `document.querySelector( "#logo" or ".logo");` للوصول لعنصر باستخدام الـ (id/ class).
- `document.querySelectorAll( "#logo" or ".logo");` للوصول لعنصر (عناصر) باستخدام الـ (id/ class).
- myElement.**children** => للوصول للعناصر الموجودة داخل عنصر ما ...
- myElement.**children[0]** => للوصول لاول عنصر داخل عنصر ما ...
- myElement.**childNodes** => للوصول للعناصر والنص والتعليقات (الابناء) الموجودة داخل عنصر ما.
- myElement.**firstChild** => للوصول لاول **ابن** موجود داخل عنصر ما ...

- myElement.**lastChild** => للوصول لاخر **ابن** موجود داخل عنصر ما ...
- myElement.**firstElementChild** => للوصول لاول **عنصر** موجود داخل عنصر ما ...
- myElement.**lastElementChild** => للوصول لاخر **عنصر** موجود داخل عنصر ما ...
- ElementOne.**prepend(**ElementTwo **or** "string"**)** : لاضافة عنصر (نص) في بداية عنصر...
- ElementOne.**append(**ElementTwo **or** "string"**)** : لاضافة عنصر(نص) في نهاية عنصر...
- ElementOne.**before(**ElementTwo **or** "string"**)** : لاضافة عنصر(نص) قبل عنصر...
- ElementOne.**after(**ElementTwo **or** "string"**)** : لاضافة عنصر(نص) بعد عنصر...
- Element.**remove()** : لازالة (حذف) عنصر...
- Element.**nextSibling** : للوصول للابن التالي (بعد العنصر المحدد)...
- Element.**nextElementSibling** : للوصول للعنصر التالي (بعد العنصر المحدد)...
- Element.**previousSibling** : للوصول للابن السابق (قبل العنصر المحدد)...
- Element.**previousElementSibling** : للوصول للعنصر السابق (قبل العنصر المحدد)...
- Element.**parentElement** : للوصول للعنصر الاب (للعنصر المحدد)...
- **document.**title; للوصول لل page title .. والذي يظهر في browser tab.
- **document.**documentElement; للوصول للعنصر الـ document الموجود به كل العناصر والخواص .
- **document.**body; للوصول لل body element .
- **document.**forms; للوصول لل form elements .
- **document.**links; للوصول لل link elements .

لاحظ: يمكن الوصول الي اول عنصر form (فقط) من خلال ... **document.**forms[0]; (وكذلك في باقي المحددات السابقة)

لاحظ: يمكن الوصول لقيمة اي سمة في العنصر من خلال .. **document.**forms[1]**.href ;** (وكذلك في باقي المحددات السابقة)

- للوصول لمحتوي الـ html الموجود داخل عنصر ما : myElement.**innerHTML**
- للوصول للمحتوي النصي (فقط) الموجود داخل عنصر ما : myElement.**textContent**

myElement.**innerHTML** = " Ibrahim Abdelaty Deefallah";

document.images[0].**id** = " pic"; \\ لتغيير او تعيين id

document.images[0].**className** = " pic"; \\ لتغيير او تعيين class

- لانشاء سمة (Attribute) : **let** myAttr = **document.createAttribute("data-custom");**

ولاضافة السمة فارغة (بدون قيمة) لعنصر ما ... myElement.**setAttributeNode(myAttr);**

- للوصول لقيمة سمة في عنصر ما : myElement.**getAttribute("href");**

- انشاء سمة جديدة وتعيين قيمة لها في عنصر ما او تغيرها اذا كانت موجودة:

myElement.**setAttribute("href" , " https:// ....."**);

- لازالة سمة معينة من عنصر ما: element.**removeAttribute("id");**

- للوصول لكل السمات الموجودة في عنصر ما : element.**attributes**

- لمعرفة هل سمة معينة موجودة في عنصر ما ام لا :

myElement.**hasAttribute("href") ;** \\ true or false

- لمعرفة هل العنصر يحتوي علي سمات (اي سمات) ام لا :

myElement.**hasAttributes() ;** \\ true or false

- لانشاء عنصر ما : **let** myElement = **document.createElement("div");**

ولاضافة العنصر (الابن) في (نهاية)عنصر الـ bady مثلا ...

document.bady.appendChild(myElement);

- لانشاء محتوي نصي : **let** myText = **document.createTextNode("data-custom");**

ولاضافة المحتوي النصي لعنصر ما ... myElement.**appendChild(myText);**

- لانشاء تعليق :

let myCom = **document.createComment(" this is comment");**

ولاضافة التعليق لعنصر ما ... myElement.**appendChild(myCom);**

Events

- **window.onload** : عند تحميل الصفحة ...
- **window.onscroll** : عند الاسكرول في الصفحة ...
- **window.onresize** : عند التغيير في الصفحة (التكبير / التصغير) ...
- myElement.**onclick** : عند الضغط علي العنصر بزر الفأرة الايسر

myElement.**onClick** = **function( ) {** Block of code **}**

- myElement.**oncontextmenu** : عند الضغط علي العنصر بزر الفأرة الايمن
- myElement.**onmouseenter** : عند الوصول للعنصر بمؤشر الفأرة
- myElement.**onmouseleave** : عند مغادرة العنصر بمؤشر الفأرة
- myElement.**onfocus** : عند الوقف علي حقل الادخال ...
- myElement.**onblur** : عند مغادرة حقل الادخال...
- myElement.**onsubmit** : عند ارسال الـ form بالضغط علي submit ...
- myElement.**focus** : لعمل فوكس علي حقل الادخال ...
- myElement.**blur** : عكس السابق ...
- myElement.**click** : للضغط علي العنصر ...

لاحظ: 1- يمكن كتابة الـ Events في ملف الـ html ..

2-

```
document.links[1].onclick = function ( event ) {
event.preventDefault()};
```

في المثال السابق: الـ event تعود علي onclick. .. و الـ preventDefault() لايقاف السلوك التلقائي للعنصر (لا يفتح اللينك عند الضغط عليه)

- myElement.**classList**.contains(“logo”) : لمعرفة هل class معين موجود في العنصر ام لا ...
- myElement.**classList**.item(“1”) : لمعرفة class موجود في عنصر من الفهرس الخاص به ...
- myElement.**classList**.add(“one” , ...) : لاضافة class (او اكثر)الي عنصر ما ...

- myElement.classList.remove("one" , ...) : لازالة class (او اكثر) من عنصر ما ...
- myElement.classList.toggle("logo") : اذا كان الـ class موجود يزيله ... واذا لم يكن موجود يضيفه.
- myElement.style.color : للوصول لخاصية css مطبقة علي عنصر (وتغييرها او اضافتها) ...

لاحظ: اذا كانت الخاصية مكونة من اكثر من كلمة تكتب بالـ camel case: backgroundColor

- myElement.style.cssText : للوصول لخواص الـ css المطبقة علي عنصر (وتغييرها او اضافتها) ...

myElement.style.cssText = "font-size:12px ; color:green;"

- myElement.style.setProperty("color", "40px", "important") : لاضافة خاصية الي الـ (inline style).
- myElement.style.removeProperty("color") : لازالة خاصية من الـ (inline style)...

لاحظ: لازالة خاصية من الـ styleSheet ....

```
document.styleSheets[0].rules[0].style.removeProperty("color")
```

- Element.cloneNode(false or true) : لعمل نسخة من العنصر .. اذا كانت القيمة false (الافتراضية) لا يأخذ نسخة من العناصر الموجودة داخل العنصر ... واذا كانت true يأخذ نسخة من العنصر بالعناصر الداخلية.
- لتطبيق action علي عنصر عن حدوث event ...

```
myElement.addEventListneer("click" , myFunction );
```

الفرق بين .addEventListneer() وبين الـ events السابقة انها ..

1. يمكن استخدامها اكثر من مره لتطبيق events من نوع واحد او مختلفين (او action) علي نفس العنصر..
2. يمكن استخدامها علي عنصر لم يتم انشاءه بعد (غير موجود) عكس الـ events السابقة التي ترجع خطأ..

لاحظ: 1) في هذا المثال التالي: الـ e.target تعبر عن العنصر الذي يتم الضغط عليه الان ...

```
document.addEventListneer("click" , function(e) {
    console.log(e.target);
    console.log(e.currentTarget.dataset.color) });
```

2) و e.currentTarget.dataset.color للوصول الي لون العنصر الذي يتم الضغط عليه ...

Browser Object Model (BOM)

- window object is the browser window
- window contain the document object
- All global variables & objects & functions are members of window object

- **window.alert(**"message" **)** or **alert(**"message"**)**
تستخدم لتظهر رسالة تحذير في المتصفح ... (هناك بدائل لها افضل .. عبارة عن مكتبات مثل: sweetalert2)

- **window.confirm(**"message" **)** or **confirm(**"message"**)**
تستخدم لتظهر رسالة تأكيد في المتصفح ... تعيد : true / false ... (هناك بدائل لها افضل)

- **window.prompt(**"message" **)** or **prompt(**"message"**)**
تستخدم لتظهر رسالة بها حقل ادخال لجمع بيانات معينة من المستخدم ... (هناك بدائل لها افضل)

- **window.open(**"URL"[opt] , "_blank" or "_self"[opt] , "width=400, .." [opt]**)** لفتح نافذة جديدة.

- **window.close( )** لاغلاق نافذة (المفتوحة بواسطة الـ open() فقط)...

- **window.stop( )** لايقاف تحميل الصفحة (النافذة) ... تكون واضحة في الصفحات كبيرة الحجم.

- **window.print( )** لطباعة الصفحة.

- **window.focus( )** لعمل فوكس علي النافذه (تستخدم في حال وجود اكثر من نافذة).

- **window.scrollTo(**500 , 400 **)** لعمل اسكرول (الي نقطة معينة تحدد من الـ x و y).

- **window.scrollBy(**500 , 400 **)** لعمل اسكرول (زيادة الاسكرول كل مرة يتم الاستخدام).

لاحظ: **window.scrollTo({** top:400 , behavior: "smooth" **});**

- **window.scrollX** للوصول الي مقدار الاسكرول علي المحور x ...
- **window.scrollY** للوصول الي مقدار الاسكرول علي المحور y ...

- **history.length( )** لمعرفة عدد العناصر الموجودة في الـ history (بما فيها المفتوحة حاليا)...
- **history.back( )** للرجوع خطوة للوراء في الـ history...
- **history.forward( )** للذهاب خطوة للأمام في الـ history ...
- **history.go(** ... -1 or 0 or 1 ... **)** للانتقال في الـ history ...

- **setTimeout(** myFunction , Millseconds , func params [opt] **)**
تستخدم في حالة اردت تطبيق دالة بعد وقت معين من تحميل الصفحة في المتصفح ...
let handler = **setTimeout(** .... **)**

- **clearTimeout(** handler**)**
تستخدم في ازالة تطبيق setTimeout من علي الدالة ...

- **setInterval(** myFunction , Millseconds , func params [opt] **)**
تستخدم في حالة اردت تكرار تطبيق دالة كل وقت معين ...

- **clearInterval (** handler**)**
تستخدم في ازالة تطبيق setInterval من علي الدالة ...

Location Object

هو كائن موجود داخل الـ window object للتحكم في الـ URL .....

- **location.href** للوصول الي الـ URL للصفحة المفتوحة..... (او تغيير الصفحة بتغيير الرابط)

- **location.host** للوصول الي الـ host (google.com) و الـ port (8080)للصفحة المفتوحة.....

- **location.hostname** للوصول الي الـ host (google.com) للصفحة المفتوحة.....

- **location.protocol** للوصول الي البروتوكول (http or https) للصفحة المفتوحة.....

- **location.hash** للوصول الي الـ hash (id) (https://chrome.google.com/#servces) .....

- **location.reload()** لاعادة تحميل الصفحة المفتوحة.....

- **location.replace(**"URL"**)** الذهاب لصفحة اخرى غير المفتوحة (الصفحة القديمة لا تظهر في الـ history)

- **location.assign(**"URL"**)** الذهاب لصفحة اخرى غير المفتوحة (الصفحة القديمة تظهر في الـ history)

- **لتخزين قيمة في كائن الـ Local Storage ... (تبقي محفوظة حتي عند اغلاق المتصفح)...**

لاضافة خاصية :

- **window.localStorage.setItem(** "color" , "red" **)**
- **window.localStorage**.fontWeight = "bold"
- **window.localStorage[**fontSize**]** = "20px"

للوصول الي خاصية :

- **window.localStorage.getItem(** "color" **)**
- **window.localStorage**.fontWeight

- `window.localStorage[fontSize]`

للوصول الي الـ key الخاص بخاصية معينة من خلال فهرسه داخل الـ Local Storage :

- `window.localStorage.key( 0 )`

لازالة خاصية :

- `window.localStorage.removeItem( "color" )`

لازالة كل الخواص :

- `window.localStorage.clear( )`

- لتخزين قيمة في كائن الـ Session Storage ... (تبقي محفوظة مادام المتصفح لم يغلق)...

لاضافة خاصية :

- `window.sesstionStorage.setItem( "color" , "red" )`
- `window.sesstionStorage.fontWeight = "bold"`

لاحظ: يمكن استخدام نفس الـ methods السابقة المستخدمة مع (Local Storage) هنا ...

لاحظ: للوصول للعنصر الذي له لون احمر مخزن في متغير ....

```
document.querySelector( `[data-color="${myColor}"]` );
```

Spread Syntax

```
myArray = [1, 2, 2, 1, 3, 4, 5];
myObj1 = {a:1, b:2};
myObj2 = { c:3, d:4};
console.log(..."Ahmed");          => A  h  m  e  d
console.log(...myArray);          => 1 2 2 1 3 4 5
console.log( [..."Ahmed"] );      => [A, h, m, e, d ]
console.log( { ... myObj1 , ... myObj2});     => { a:1, b:2, c:3, d:4 }
```

لاحظ: الـ Spread Operator لا يغير في الـ (Array / Object / ..) الرئيسية ...

Regular Expression

- input.match(Pattern)
  - Matches a string against a regular expression pattern
  - Returns an array with the matches
  - Returns null if no match is found

```
myString = "My name is Ibrahim Abdelaty : ibrahim";

console.log(myString.match( /Ibrahim/ ) );    => ["Ibrahim" , index: 11 ]
```

Modifiers (Flags) :

- i => case insensitive
- g => global (All)
- m => multi lines

```
console.log(myString.match( /ibrahim/i ) );    => ["Ibrahim" ]

console.log(myString.match( /ibrahim/ig ) );    => ["Ibrahim" , "ibrahim" ]
```

Ranges :

- (x|y) => x or y
- [0-9] => 0 To 9
- [^0-9] => Any char not 0 To 9
- [a-z] => a To z
- [^a-z] => Any char not a To z
- [A-Z] => A To Z
- [^A-Z] => Any char not A To Z
- [abc] => a and b and c
- [^abc] => not a and b and c

```
console.log(myString.match( /(name|ibrahim)/i ) );    => ["name"]

console.log(myString.match( /[A-Z]/g));    => [ M , I, A]
```

Character Classes :

- . => Any character expect newline or any character terminators.
- \w => Word characters [a-z , A-Z , 0-9 , _].
- \W => Non word characters.
- \d => Digits characters from 0 to 9.
- \D => Non digits characters.

- \s => Whitespace.
- \S => Non whitespace.
- \b => At the (beginning or end) of a word.
- \B => Non at the (beginning or end) of a word.

```
console.log(myString.match( /\bibr/ig));     => [ Ibr , ibr]
console.log(myString.match( /him\b/ig));    => [ him , him]
```

Quantifiers :

- n+ => One or More.
- n* => Zero or More.
- n? => Zero or One.
- n{x} => Number of. \d{5} (five digits)
- n{x,y} => Range.
- n{x, } => At least x.
- n$ => End with something (true / false).
- ^n => Start with something (true / false) .
- (?=n) => Followed by something.
- (?!n) => Not followed by something.

```
let emails = "ibrahim@gmail.com  wael@@yahoo.com  google.com";
console.log(emails.match( /\w+@\w+.(com|org)?/ig ));  => ['ibrahim@gmail.com']
console.log(emails.match( /\w+@\w+(?=.com)/ig ));  => ['ibrahim@gmail']
```

- Pattern.test(input) لمعرفة هل يلتزم الـ input بالـ Pattern ام لا (ترجع true او false)...

```
console.log( /(\bhim|him\b)/i.test("ibrahim") );   => true
```

لاحظ: لتخطي احد الحرواف الخاصة نستخدم (\ ) ...

- لتبديل حرف (كلمة) ... بحرف (كلمة)...

```
let text = "this is @ , this is @";
console.log(text.replace( "@" , "JS"));      => "this is JS , this is @"
console.log(text.replaceAll( "@" , "JS"));  => "this is  JS , this is JS"
```

لاحظ: يمكن استخدام ... text.replaceAll(Pattern , "New text");

لاحظ: يمكن كتابة ...(اقل شيوعا) ... new RegExp("Pattern" , "Modifier(s)");