انواع دیتا تایپ

- Number : برای نگهداری اعداد استفاده میشود.
- String : برای نگهداری رشته ها استفاده میشود.
- Boolean : برای متغیرهای منطقی صفر و یک استفاده میشود.
- Undefined : معنای آن این است که این متغیر تعریف شده است اما هنوز مقداری به آن داده نشده است.
- Null : اما null خود مقداریست که به متغیری assign میشود و معنای آن هیچ است.

ویژگی هایی اصلی که میبایست در نام متغیر به آن دقت کرد:

- با عدد شروع نمیشود.
- با _ یا $ یا حروف میتوان شروع کرد.
- در میان اسم نمیتوانیم / بگذاریم.
- در میان اسم نمیتوانیم – بگذاریم.
- نمیتوان از کلمات رزرو شده استفاده کرد.

در جاوا اسکریپت دو ویژگی Coercion و Mutate است که در حالت اول به اجبار تایپی را به یک متغیر داده میشود ودر حالت دوم تایپ متغیر تغییر میکند :

```
/*****************************
* Variable mutation and type coercion
*/
var firstName = 'John';
var age = 28;
// Type coercion
console.log(firstName + ' ' + age);
var job, isMarried;
job = 'teacher';
isMarried = false;
console.log(firstName + ' is a ' + age + ' year old ' + job + '. Is he married? ' + isMarried);
// Variable mutation
age = 'twenty eight';
job = 'driver';
alert(firstName + ' is a ' + age + ' year old ' + job + '. Is he married? ' +
isMarried);
var lastName = prompt('What is his last Name?');
console.log(firstName + ' ' + lastName);
```

میدانیم که به کمک دستور زیر میتوان type یک متغیر را متوجه شد :

```
typeof ("slm") ;
```

به عنوان مثال خروجی عبارت بالا مقدار String است.

اولویت اجرای دستورات و عبارات را میتوانید با کلیک بر اینجا مشاهده کنید.

آشنایی با ساختار if و جبر Boolean :

```
var firstName = 'John';
var age = 20;
if (age < 13) {
   console.log(firstName + ' is a boy.');
} else if (age >= 13 && age < 20) {
   console.log(firstName + ' is a teenager.');
} else if (age >= 20 && age < 30) {
   console.log(firstName + ' is a young man.');
} else {
   console.log(firstName + ' is a man.');
}
```

میدانیم برای and میبایست از && و برای or میبایست از || استفاده کرد.

آشنایی با ساختار switch و case :

```
var job = 'instructor';
switch (job) {
  case 'teacher':
  case 'instructor':
     console.log(firstName + ' teaches kids how to code.');
     break;
  case 'driver':
     console.log(firstName + ' drives an uber in Lisbon.');
     break;
  case 'designer':
     console.log(firstName + ' designs beautiful websites.');
     break;
  default:
     console.log(firstName + ' does something else.');
}
```

میدانیم اگه چندیدن case بدون break بگذاریم دستورات موجود تا اولین break برای همه case ها اتفاق میافتد؛ ضمنا میدانیم که هنگامی که هیچ کدام از case ها برقرار نباشد به سراغ اجرای breack میرود.

ضمنا میتوانیم روی case ها عملیات منطقی انجام بدهیم. به تکه کد زیر توجه کنید :

```
age = 56;
switch (true) {
  case age < 13:
    console.log(firstName + ' is a boy.');
    break;
  case age >= 13 && age < 20:
    console.log(firstName + ' is a teenager.');
    break;
  case age >= 20 && age < 30:
    console.log(firstName + ' is a young man.');
    break;
  default:
    console.log(firstName + ' is a man.');
}
```

حال میخواهیم با ternary operator آشنا شویم

```
condition ? instruction1 : instruction2 ;
```

اگر شرایط برقرار بود instruction1 انجام میپذیرد و در غیر این صورت instruction2 رخ میدهد.

حال به سراغ دو تعریف مهم میرویم. Fulsy value به مقادیری گفته میشود که چنانچه در condition قرار بگیرد آن شرط غلط حساب میشود و وارد دستورات نمیشود. Fulsy value ها عبارتند از :

- “”
- 0
- Null
- Undefined
- False
- not a number : NaN

دیگر عبارت ها اگر در condition قرار بگیرند آن شرط pass شده و دستورات بعد از آن اجرا میشود.

میخواهیم به بیان تفاوت == و === بپردازیم. اگر برابری ای با === بررسی شود. برابری تایپ دو متغیر نیز بررسی میشود. اما چنانچه == استفاده شود فقط مقادیر مورد بررسی قرار میگیرد.

```
if("1" == 1)
  console.log("i'm here")
```

مثلا وقتی این را بنویسیم در کنسول مقدار I'm here چاپ میشود.

امیدوارم با مفهوم function آشنا باشید. در این صورت میدانیم که function برای عمل کردن به اصل don't try yourself یا همان DRY استفاده میشود. سه نوع تعریف تابع داریم :

نوع اول که به آن function declaration میگویند و به این شکل است :

```
function calculateAge(birthYear) {
  return 2018 - birthYear;
}
```

نوع دوم که به آن function expression میگویند و به شکل زیر است :

```
var whatDoYouDo = function(job, firstName) {
  switch(job) {
    case 'teacher':
      return firstName + ' teaches kids how to code';
    case 'driver':
      return firstName + ' drives a cab in Lisbon.'
    case 'designer':
      return firstName + ' designs beautiful websites';
    default:
      return firstName + ' does something else';
  }
}
```

نوع سوم که کاربرد آن برای وقت هایی است که نمیخواهیم دوباره از تابع استفاده کنیم(!)

```
function (){
  // do somthings
}
```

نحوه تعریف آرایه ها برای ذخیره ی دسته ای از داده ها استفاده میشود که بهتر است این داده ها از یک data type باشد ولی در جاوا اسکریپت محدودیتی برای این موضوع نیست و در هر آرایه میتوان انواع data type ها را جا داد.

برای تعریف آرایه مطابق زیر عمل میکنیم :

```
// Initialize new array
var names = ['John', 'Mark', 'Jane'];
var years = new Array(1990, 1969, 1948);
console.log(names[2]);
console.log(names.length);
var array = new Array() ;
console.log(array)
// Mutate array data
names[1] = 'Ben';
names[names.length] = 'Mary';
console.log(names);
names[7] = 'cristian'
console.log(names[6])
names.push('mahdi');
console.log(names.pop())
names.pop()
console.log(names)

// Different data types
var john = ['John', 'Smith', 1990, 'designer', false];
john.push('blue');
john.unshift('Mr.');
console.log(john);
john.pop();
john.pop();
john.shift();
console.log(john);
console.log(john.indexOf(23));
var isDesigner = john.indexOf('designer') === -1 ? 'John is NOT a designer' : 'John IS a designer';
console.log(isDesigner);
```

خب حال میبایست راجع به تکه کد بالا نکاتی را ذکر کنیم. اگر مثلا در آرایه ای که سه عنصر داشته باشد. عنصر خانه ۵ ام را مقدار دهی کنیم خانه های شماره 0,1,2,5 مقدار دهی شده است و خانه های شماره 3,4 ساخته میشوند ولی مقدار دهی نمیشوند. و مقدار فعلی آن ها undefined میباشد. چنانچه خانه 5 را هم حذف کنیم باز مقدار خانه ها ۳ و ۴ همان مقدار قبلی را نگه میدارد.

دستور push به آخرین خانه آرایه اضافه میکند.

دستور pop آخرین خانه آرایه را حذف میکند.

دستور unshift اضافه کردن عنصر به ابتدای آرایه.

دستور shift حذف عنصر اول.

دستور indexOf که ورودی میگیرد و اولین جایی که ورودی را در آرایه پیدا کند به عنوان خروجی برمیگرداند و اگر پیدا نکرد -۱ را برمیگرداند.

حالا اگر بخواهیم از چند نوع مختلف data type در کنار هم داشته باشیم و این اطلاعات متفاوت باشند ولی به هم مربوط باشند میبایست از object استفاده کرد. نحوه تعریف object را میبینیم :

```
// Object literal
var john = {
  firstName: 'John',
  lastName: 'Smith',
  birthYear: 1990,
  family: ['Jane', 'Mark', 'Bob', 'Emily'],
  job: 'teacher',
  isMarried: false
};
console.log(john.firstName);
console.log(john['lastName']);
var x = 'birthYear';
console.log(john[x]);
john.job = 'designer';
john['isMarried'] = true;
console.log(john);
// new Object syntax
var jane = new Object();
jane.firstName = 'Jane';
jane.birthYear = 1969;
jane['lastName'] = 'Smith';
console.log(jane);
```

میتوان به یک آبجکت function هم اضافه کرد که به آن method میگویند برای اضافه کردن یک متد به object میبایست از object expression استفاده کرد :

```
var john = {
  firstName: 'John',
  lastName: 'Smith',
  birthYear: 1992,
  family: ['Jane', 'Mark', 'Bob', 'Emily'],
  job: 'teacher',
  isMarried: false,
  calcAge: function() {
    this.age = 2018 - this.birthYear;
  }
};
john.calcAge();
console.log(john);
```

ایجاد حلقه در جاوا اسکریپت هم به صورت زیر است:

```
var john = ['John', 'Smith', 1990, 'designer', false, 'blue'];
for (var i = 0; i < john.length; i++) {
  console.log(john[i]);
}
// While loop
var i = 0;
while(i < john.length) {
  console.log(john[i]);
  i++;
}
// continue and break statements
var john = ['John', 'Smith', 1990, 'designer', false, 'blue'];
for (var i = 0; i < john.length; i++) {
  if (typeof john[i] !== 'string') continue;
  console.log(john[i]);
}
for (var i = 0; i < john.length; i++) {
  if (typeof john[i] !== 'string') break;
  console.log(john[i]);
```

```
}
// Looping backwards
for (var i = john.length - 1; i >= 0; i--) {
  console.log(john[i]);
}
```