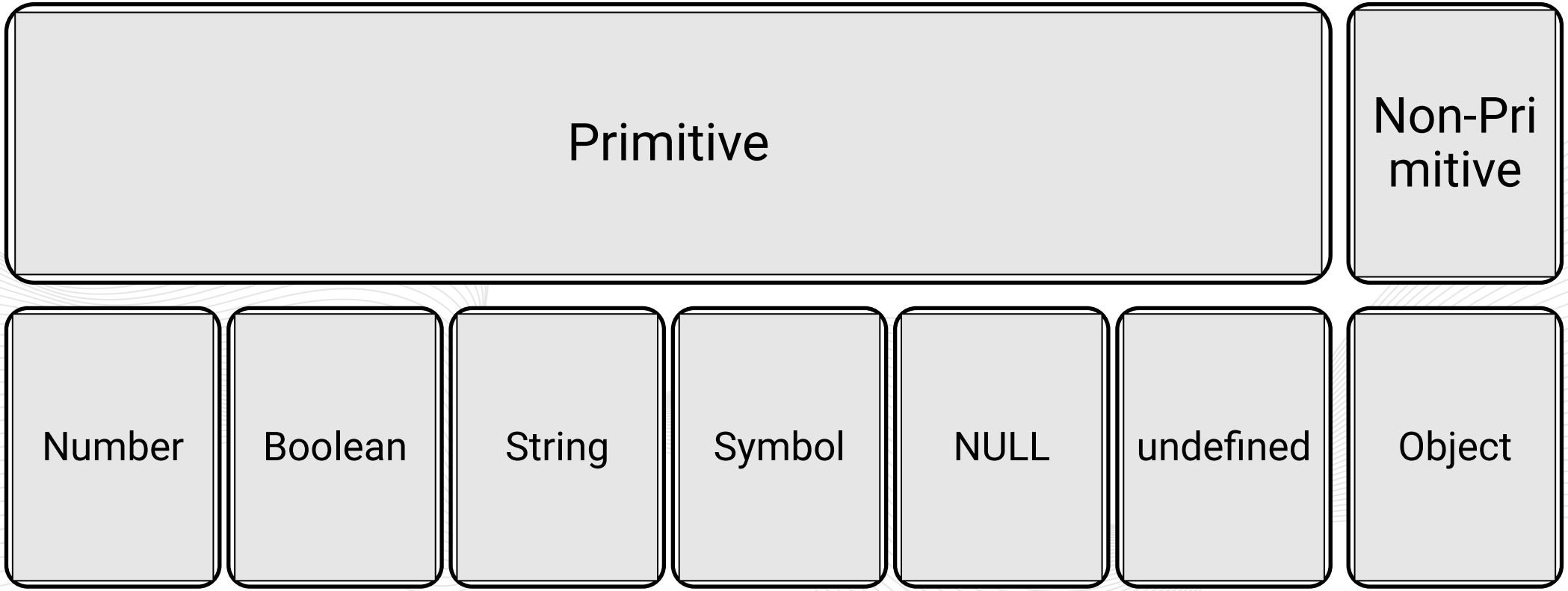
Типове данни в JavaScript

Съдържание

Data Types



01 Примитивни типове данни

number

Типът number може да бъде:

- цяло число;
- число с плаваща запетая;
- експоненциален запис;
- ‘NaN’;
- ‘Infinity’.

```
var a = 250; //integer value
var b = 25.5; //floating
    //point value
var c = 10e4; //exp. value
console.log(typeof(a));
console.log(‘hi’ * 5); //NaN
console.log(typeof(Nan));
console.log(Number.MIN_VALUE);
console.log(Number.MAX_VALUE);
console.log(42 / +0); //Inf.
console.log(42 / -0); //-Inf.
console.log(0 === -0) //true
```

bigint

Типът bigint е числов примитивен тип, който представя по-големи цели числа. Число от тип bigint може да се създаде като се добави “n” в края на цяло число.

```
const value1 =
90071992512474099 8n;
const result1 = value1 + 1n;
console.log(result1);
console.log(typeof(result1));

const x =
BigInt(Number.MAX_SAFE_INTEGER);
console.log(x + 1n === x + 2n);

const value2 =
90071992512474099 8n;
const result2 = value2 + 1;
console.log(result2); //Error!
```

boolean

Логическият тип boolean приема само две стойности – true и false.

```
console.log(typeof(true));
//return boolean
console.log(typeof(false));
//return boolean

const bool = false;
if(bool){
   console.log(“Boolean
cond. is true”);
}
else{
   console.log(“Boolean
cond. is false”);}
```

string

String (низ) е поредица от символи и се използва за представяне на текст. Може да се представи по три начина:

- единични кавички;
- двойни кавички;
- чрез backticks.

```
//strings example
const name = ‘ram’;
const name1 = “hello”;
const result = `The names are
${name} and ${name1}`;

console.log(typeof(name));
console.log(typeof(name1));
console.log(typeof(result));
console.log(typeof(“5” == 5));
console.log(typeof(“5” === 5));
```

Достъп до символ на string

- Да „третираме“ string-а като масив:

```
const a = ‘hello’;
console.log(a[1]); //e
```

- Да използваме метода charAt():

```
const a = ‘hello’;
console.log(a.charAt(1)); //e
```

Основни методи на string

- replace(…);
- split(…);
- slice(start, end);
- search(…);
- toUpperCase() && toLowerCase();

```
const str = “I like to eat apples”;
const newStr = str.replace(‘apples, ‘oranges’);
console.log(newStr);
const newStr1 = str.split(‘like to‘);
console.log(newStr1);
const newStr2 = str.slice(2, 6);
console.log(newStr2);
const pos = str.search(‘like’);
console.log(pos);
const lowerStr = str.toLowerCase();
const upperStr = str.toUpperCase();
```

symbol

Symbol:

- уникален и с постоянна стойност;
- той може да се използва за добавяне на свойство на обект.

```
const value1 = Symbol('hello');
const value2 = Symbol('hello');
console.log(value1);
//Symbol(hello)

console.log(value1 === value2);
//false

let id = Symbol('id');
let student = {
   name: "John Doe",
   [id]: 77777 //adding symbol
};
console.log(student);
```

undefined

Примитивният тип undefined представлява стойност, която не е дефинирана. Можем да я получим когато:

- имаме само декларирани променливи, без да им се дава начална стойност;
- достъпим свойство, което го няма в обект;
- достъпим елемент от масив чрез индекс, който е извън границите на масива.

```
let value;
console.log(value); //undefined
let value1 = undefined;
console.log(value1); //undefined

const person = {name: “John”};
console.log(person.egn);

const arr = [1,2,3];
console.log(arr[3]);
```

null

Null представлява умишлено отсъствие на каквато и да е ст-ст на обект.

```
var f = null;
console.log(f);

function func(param){
   if(param === null){
      console.log(“null val”);
   }
   else{
      console.log(“not null val”);
   }
}
func(null);
func(‘Hello’);
```

null vs undefined

```
console.log(typeof(undefined)); //undefined

console.log(typeof(null)); //object!

console.log(null == undefined); //true

console.log(null === undefined); //false

console.log(null === null); //true

console.log(undefined == undefined); //true
```

02 Обекти и масиви

Objects (1)

Непримитивен референтен тип данна, състоящ се от множество двойки ключ-стойност

```
let user = {};
let user = new Object();
let user = {name: “John”,
            age: 30
            “has id”: true};
-----------------------------
user.name;    // “John”
user[“name”];
-----------------------------
user.admin = true;
user[“admin”] = true;
-----------------------------
delete user.admin;
delete user[“admin”]
```

Objects (2)

Променливите от тип обект са референции към обекта.

```
let a = {};
let b = {};
let c = a;

a == b;  // false
a === b; // false

a == c;  // true
a === c; // true
-------------------------------
const a = {};
a.newProp = true // ok
a = {};          // error
```

Objects (3)

Копиране

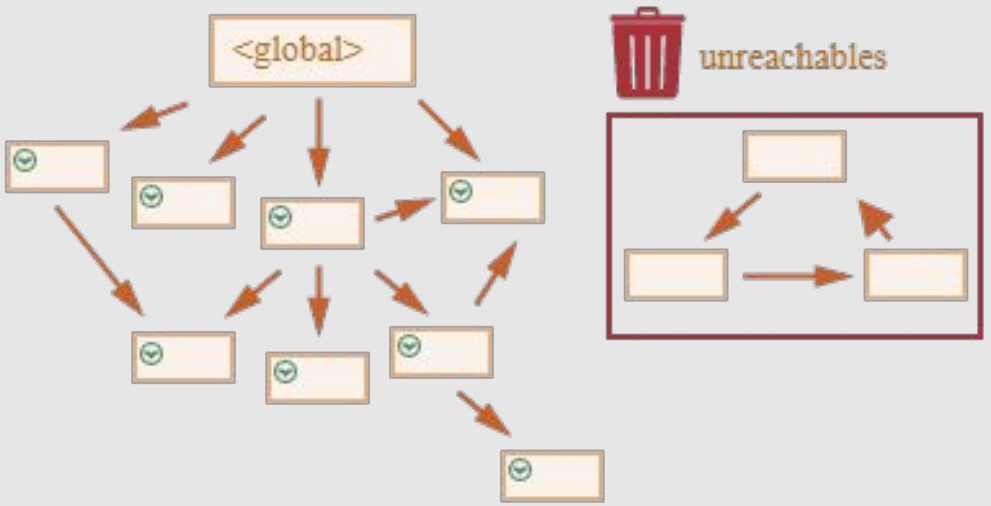
```
// shallow:

let user1 = {name: “John”};
let user2 = user1;

delete user1.name;
user2.name;     // undefined
```

```
// Deep copy:

Object.assign(dest,src1,src2);
dest = Object.assign({},src1,src2);

dest = structuredClone(src);

dest.cloneDeep(src); // lodash lib
```

Garbage collector

Достъпни стойности:

- Функциите, незавършили своето изпълнение заедно с техните променливи и параметри
- Глобални променливи
- Системни “вътрешни” променливи
- Всяка стойност, която може да се достъпи чрез горепосочените



Methods

Метод на обект наричаме всяка функция, реферирана от негов ключ.

```
let user = {};
user.sayHi = function(){alert(“Hi”);};
-----------------------------------------
function sayHi() {alert(“Hi”);};
user.sayHi = sayHi;
-----------------------------------------
let user = {
        sayHi : function(){alert(“Hi”);}
           }
-----------------------------------------
let user = { sayHi(){alert(“Hi”);}};
-----------------------------------------
user.sayHi() // “Hi” msg will pop up
```

Constructor

Функция, която следва следните конвенции:

1. започва с главна буква
2. извиква се само с оператора new

```
function User(name,age){
    // this={};
        this.name = name;
        this.age  = age;
    // return this;
 }

let user = new User(“John”, 30);
```

Conversion to Primitive

Съществува преобразуване
само към число и стринг

В булев контекст стойността
на обект е винаги true

```
[Symbol.toPrimitive](hint){
  return
hint==”string” ? this.name : this.age;}

valueOf() { return this.age;}

toString() { return this.name;}


alert(user)     // hint is string
alert(+user)    // hint is number
alert(user+100) // hint is default
```

Properties

Свойствата имат 3 флага:

writable
enumerable
configurable

```
Object.getOwnPropertyDescriptor(user,"name")
```

```
Object.defineProperty(user,"name",
                      {value:"John"});
```

get & set methods

Извикват се при четене или промяна на стойността на дадено свойство

```
let user={name: ”John”,
          age: 30,
          get info(){
      return this.name+this.age;
           }
           set name(name){
       this.age=name;
           }
}
user.info;   // “John30”
user.name=”Jack”;
user.age;    // “Jack”
```

Arrays

```
let arr = new Array();
let arr = [];
let arr = [“hi”,{name:”John”},true,];
-------------------------------------
arr[1].name;   // “John”
arr[-1].name;  // “John”
arr[3] = false;
arr.length;  // 3
-------------------------------------
arr.shift(); // “hi”
arr.unshift();
arr.pop();   // false
arr.push(12);
```

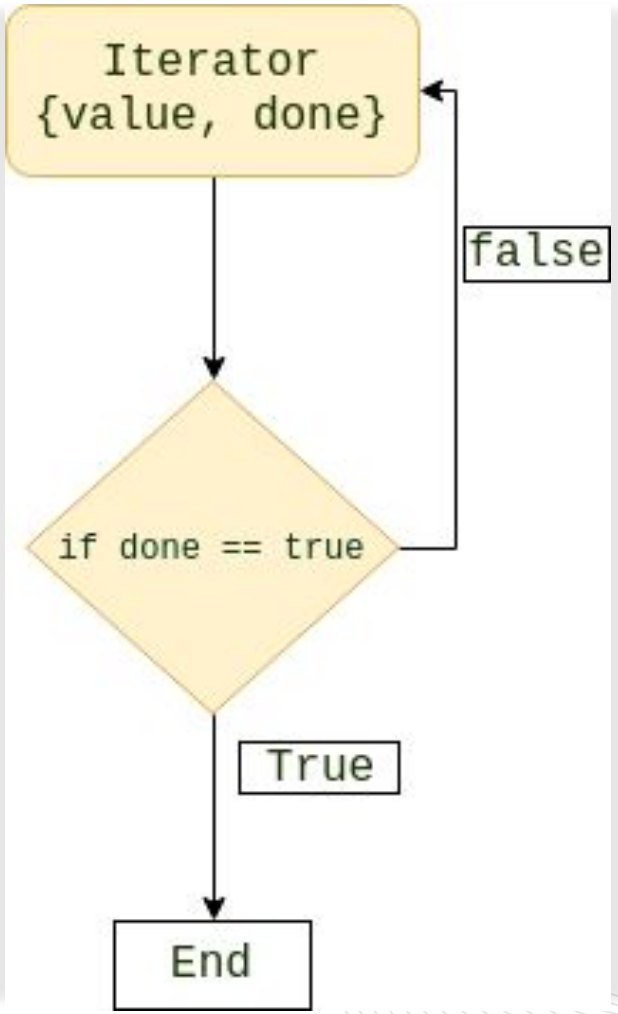
```
for(let value of arr) {alert(value);}

-------------------------------------

let arr=[];
arr[123]=”hehe”;
arr.length; // 124
arr.length=0;
arr[123]; // undefined
```

Array methods

```
let arr=[1,2,3,4];
let removed = arr.splice(0,2,9);
alert(arr); // 9,3,4
alert(removed); // 1,2
-------------------------------
arr.concat([7],7) // 9,3,4,7,7
-------------------------------
arr.indexOf(true,2) // -1
-------------------------------
arr.sort();
arr.sort(compareNumeric);
-------------------------------
arr.reverse();
```

```
“a|b|c”.split(‘|’) // [a,b,c]
“a|b|c”.split(‘’)  // [a,|,b,|,c]

-------------------------------

[a,b,c].join(‘|’)  // “a|b|c”
```

03 Итератори и генератори



Итератори

- Iterator protocol
  ➔ next()
- Iterable protocol
  ➔ [Symbol.iterator]

```
const arr = ['hi', {name:'John'},
true,];
const iterator =
arr[Symbol.iterator]();
let result = iterator.next();
while(!result.done) {
   console.log(result.value);
   result = iterator.next();
}
```

Генератори

- function* декларация
  ➔ връща Generator обект
- Generator обектът изпозлва и двата протокола iterator и iterable
- yield операторът се използва, за да спира на пауза и да пуска отново генериращата функция

```
function* generatorFunction() {
  yield 'Neo'
  yield 'Morpheus'
  yield 'Trinity'
}
const generator =
generatorFunction()
console.log(generator.next())
console.log(generator.next())
console.log(generator.next())
console.log(generator.next())
console.log(generator.next())
```

Източници(1):

Presentation Template: SlidesMania

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures?fbclid=IwAR3Lb2kiTeFNSmWkcZ_r2o7Hu4lj0RhK-bXkP8QKMngflnCPbQdqKhTYpVw#symbol_type

https://www.edureka.co/blog/data-types-in-javascript/?fbclid=IwAR2I8InCtI44s3eXbQcFM-tFRTP0IDQK65b9SyMlSawasOEgTQ4qfdhQHZU

https://www.w3schools.com/js/

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null?fbclid=IwAR3WQvM6o8QPXsOMlWem0cZMjk4vtNir_a7iPwkkNba6rrF_oesKthOU1dc

https://www.freecodecamp.org/news/javascript-nullable-how-to-check-for-null-in-js/

https://javascript.info/

Източници(2):

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Generator

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols

https://en.wikipedia.org/wiki/Yield_(multithreading)

https://www.digitalocean.com/community/tutorials/understanding-generators-in-javascript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield

https://bg.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD)

Благодарим за вниманието!

```
<p> Изготвили: Александър Ангелков, Георги Цеков, Михаела Ангелова </p>
```