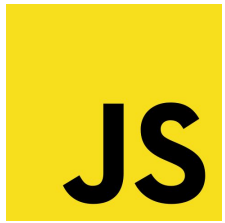
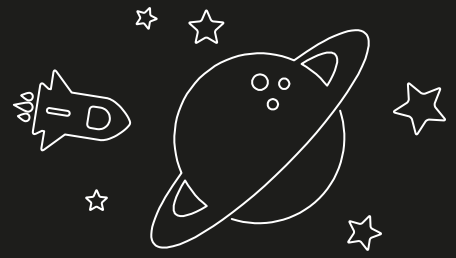
Les fondements



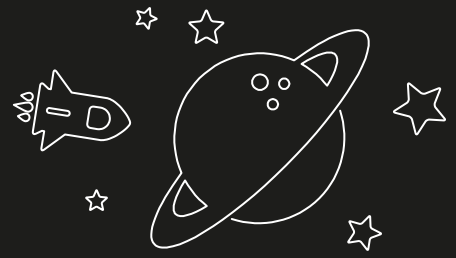


Structure

2

```
/*
 * @param input
 * @return boolean
 */
function echo(input) {
    // Display message into the console
    console.log(“message”);
    return true;
}
```

3



Types de données

4

Typage faible

6 types différents

2 catégories

4 type de déclarations possibles

Les types de données

Types primitifs

String => “message”

Number => 2 2.3

Boolean => true false

Undefined => undefined

Types complexes

Object

object => { prop1: true, prop2: “yes” }

null => null

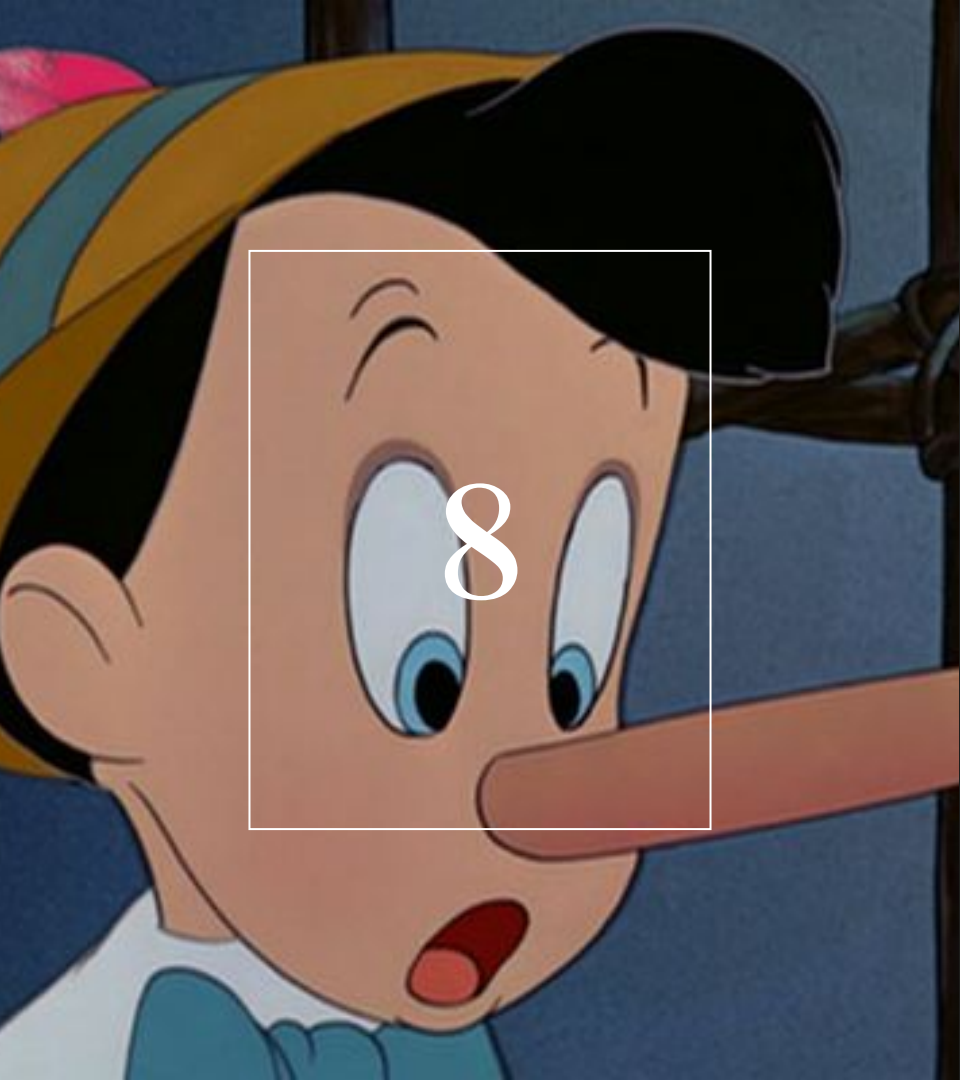
array => [1, 2, 3]

Function

“

MENSONGES !!!!

7



JAVASCRIPT EST UN LANGAGE ORIENTE OBJET A PROTOTYPE

- Chaque type est “wrappé” par un objet
- Panel de fonctions intégrées et customisables

```
// Number
1.390.toFixed(3)
// String
“message”.split()
// Boolean
true.toString()
// Array
[13, 9, 0].length
// Function
function split(arg1){}
split.arguments
```

Les opérateurs

	+	-	*	/	.
Number	Add	Sub	Mult	Div	
String	Concat				
Object					Access
Boolean	Add	Sub	Mult	Div	

Résultats inattendus

NaN

Infinity

Déclaration de variables

foo = 3

Déclaration globale

var foo = 3

Déclaration locale

let foo = 3

Déclaration au block

const foo = 3

Let + constante de référence

Attention !!! Phénomène de Hoisting

Javascript remonte automatiquement les déclarations de variables sans les initialiser en haut du scope courant pour le keyword **var**


```
var foo = 1;

(function() {
  console.log(foo);
  var foo = 2;
  var baz = 3;
  bar = 4;
})();

console.log(foo);
console.log(bar);
console.log(baz);
```

Undefined

1
4
erreur

Les closures

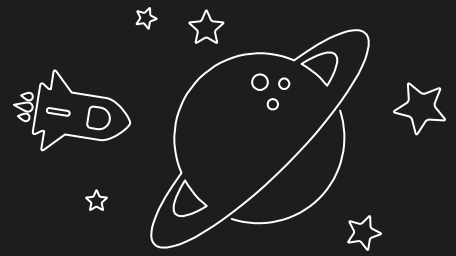
Fonction dans une fonction

Javascript se souvient de l'environnement de chaque fonction.

Une variable qui ne devrait plus exister peut donc continuer à être appelée.

```
function creerFonction() {
  var nom = "Mozilla";
  function afficheNom() {
    console.log(nom);
  }
  return afficheNom;
}

var maFonction = creerFonction();
maFonction();
```



BLOC DE STRUCTURE

13

Les boucles

Boucle for

```
for(init;cond;acc) {
    //Something
}
```

Boucle for ... in

```
for(x in object) {
    //Something
}
```

Boucle do ... while

```
do {
    //Something
} while (cond)
```

Boucle while

```
while (cond) {
    //Something
}
```

Break *X*

Stop les itérations de X boucles

Continue

Passe à l'itération suivante

Les conditions

if

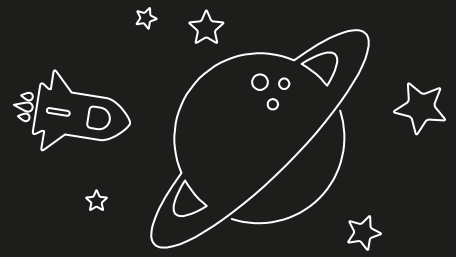
```
If (cond) {
    //Something
} else if (cond2) {
    //Something
} else {
    //Something
}
```

Opérateurs

>, >=
<, <=
==
===

switch

```
switch(var) {
  case v1:
    //Something
    break;
  default:
    //Something
    break;
}
```



Exercice 1
Manipulation de chaînes

16

./exercise-1/string.js

ucfirst

1ère lettre en MAJ

hello world => Hello world

capitalize

1ère lettre de chaque mot en MAJ

hello world => Hello World

camelCase

Capitalize + coller les mots

hello world => HelloWorld

snake_case

Joindre les mots par des underscores en MIN

leet - Cryptage (uniquement les voyelles)

anaconda => 4n4c0nd4

A=>4, E=>3, I=>1, O=>0 ,U=> (_), Y=>7

prop_access

“animal.type.name” => animal[“type”][“name”]

prop_access(prairie, “animal.type.name”)

=> praire.animal.type.name => “chien”

Si attribut non existant, afficher le chemin jusqu’à l’attribut => “animal.gender not exist”

Si path vide ou null, renvoyer l’objet complet

verlan

Inverser chaque mot d’une phrase

Hello world => olleH dlrow

yoda

Inverser la position des mots d’une phrase

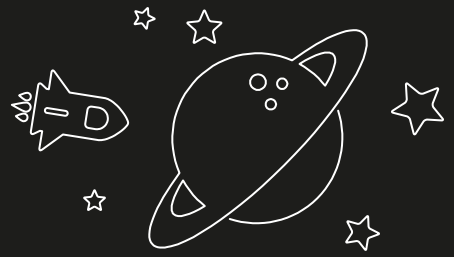
Hello world => world Hello

vig - Cryptage => Chiffre de Vigenère

wikipedia + crypto => yzixisfzy

Renvoyer une chaîne vide en cas de valeur non String

Commit : [DONE] exercice 1



Exercice 2

Jouons avec les types

Jouons avec les types ./exercice-2/type-check.js

type_check_v1

Vérifier que le type de l'arg1 correspond à l'arg2

type_check_v1(1, "number") => true

type_check_v2

Gérer un objet conf à vérifier

type: type de l'arg1

value: valeur de l'arg1

enum: valeurs possibles de l'arg1

type_check

Gérer une conf récursive

Properties: liste des propriétés de l'objet associé à sa conf

```
{
 type: "object",
 properties: {
   prop1:  { type: "number" },
   prop2: { type: "string", enum: ["val1", "val2"] },
   prop3: { type: "object", properties: { prop31: "number"
} },
   prop4: { type: "array", properties: [ "boolean"]}
 }
}
```


Commit : [DONE] exercice 2

Exemples

type_check_v1

```
type_check_v1(1, “number”) => true
```

type_check_v2

```
type_check_v2({prop1: 1}, {type: "object"})
        => true
type_check_v2("foo", {type: "string", value: "foo"})
        => true
type_check_v2("bar", {type: "string", value: "foo"})
        => false
type_check_v2(3, {enum: ["foo", "bar", 3]})
        => true
```

type_check

Gérer une conf récursive

Properties: liste des propriétés de l'objet associé à sa conf

```
{
 type: "object",
 properties: {
   prop1:  { type: "number" },
   prop2: { type: "string", enum: ["val1", "val2"] },
   prop3: { type: "object", properties: { prop31: "number"
} },
   prop4: { type: "array", properties: [ "boolean"]}
 }
}
```


Commit : [DONE] exercice 2