



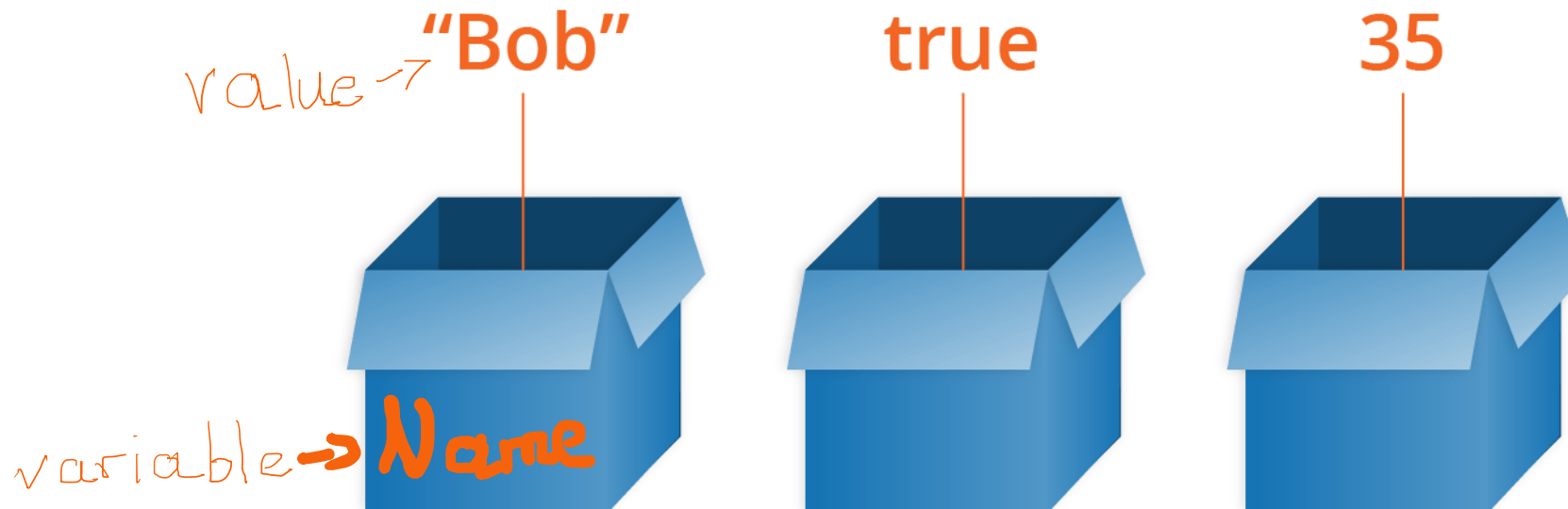
JavaScript variables

What is a variable?

- Μια μεταβλητή είναι ένα “δοχείο” για μια τιμή
 - Η τιμή αυτή θα μπορούσε να είναι ένας αριθμός που θα μπορούσαμε να χρησιμοποιήσουμε σε ένα άθροισμα
 - μια συμβολοσειρά που θα μπορούσαμε να χρησιμοποιήσουμε ως μέρος μιας πρότασης κλπ.
- Οι μεταβλητές μπορούν να περιέχουν σχεδόν οτιδήποτε — όχι μόνο συμβολοσειρές και αριθμούς.
- Οι μεταβλητές μπορούν επίσης να περιέχουν **πολύπλοκα δεδομένα** και ακόμη και ολόκληρες **συναρτήσεις**

What is a variable?

- ΠΡΟΣΟΧΗ: Λέμε ότι οι μεταβλητές περιέχουν τιμές.
- Αυτή είναι μια σημαντική διάκριση που πρέπει να γίνει. Οι μεταβλητές δεν είναι οι ίδιες οι τιμές, είναι δοχεία για τις τιμές. Μπορείτε να σκεφτείτε ότι είναι σαν κουτιά στα οποία μπορείτε να αποθηκεύσετε πράγματα.



JavaScript Datatypes

- Η JavaScript μας επιτρέπει να εργαζόμαστε με τους ακόλουθους τύπους δεδομένων
- Primitive values(primitive values είναι αμετάβλητες: δεν μπορούν να αλλάξουν, αν και στη μεταβλητή που την κατέχει μπορεί να εκχωρηθεί εκ νέου άλλη τιμή)
 - **Αριθμούς** είτε ακέραιους είτε δεκαδικούς χωρίς κάποια διάκριση & **Συμβολοσειρές**
 - **Boolean**: true/false.
 - **BigInt** (JavaScript BigInt variables are used to store big integer values that are too big to be represented by a normal JavaScript integer-> is only accurate up to 15 digits/ BigInt can not have decimals)
 - **Symbol** (const sym2 = Symbol("id"); Every **Symbol()** call is guaranteed to **return a unique Symbol**. Symbols are often used to add **unique property keys to an object** that won't collide with keys any other code might add to the object)
 - **Μηδέν & απροσδιόριστο** (null and undefined)
- complex data type: **Αντικείμενα** (objects)

JS: Untyped language

- Untyped language: μια μεταβλητή μπορεί να αποθηκεύει μια τιμή από οποιονδήποτε τύπο δεδομένων χωρίς να απαιτείται **η εκ των προτέρων δήλωση του τύπου**
- Ο τύπος της τιμής που αποθηκεύεται μπορεί να αλλάξει κατά την εκτέλεση του κώδικα και η JavaScript το διαχειρίζεται κατάλληλα

Μεταβλητές στη JavaScript

- Υπάρχουν 3ς διαφορετικές λέξεις κλειδιά για τον ορισμό μεταβλητών:
 - var
 - let
 - const

Var

- JavaScript variables: μπορούν να θεωρηθούν ως δοχεία για τα δεδομένα
- var: declaration of variables

```
<script>
```

```
    var name;
```

```
    var surname;
```

```
</script>
```

var

- Μπορούμε να **εκχωρήσουμε μια τιμή** σε μια μεταβλητή χρησιμοποιώντας τον τελεστή =
 - a. όταν την δηλώσουμε
 - b. ή μετά τη δήλωση και πριν την πρόσβαση σε αυτήν

```
var name= "Aristea";
```

```
var surname;
```

```
surname="Kontogianni";
```

Πολλές μεταβλητές μπορούν επίσης να δηλωθούν ταυτόχρονα σε μία μόνο γραμμή

```
var name= "Aristea", surname;
```


let

Λέξη – κλειδί let : υπάρχουν οι ορισμένες διαφορές σε σχέση με τη λέξη – κλειδί var. Οι μεταβλητές που ορίζονται με let:

- δεν μπορούν να δηλωθούν ξανά
- πρέπει να δηλώνονται πριν από τη χρήση
- έχουν Block Scope : η περιοχή μέσα σε ένα if, switch , for και while .
Γενικότερα όποτε βλέπουμε {curly brackets} είναι ένα block

let

Οι μεταβλητές που ορίζονται με **let** **δεν μπορούν να δηλωθούν ξανά**

```
let name = "Aristea Kontogianni";
```

```
let name = 0; //Uncaught SyntaxError: Identifier 'name' has already  
been declared
```

Or

```
var name = 0; //Uncaught SyntaxError: Identifier 'name' has already  
been declared
```

let

Οι μεταβλητές που ορίζονται με `let` πρέπει να δηλώνονται πριν από τη χρήση

```
myName = "Aristea";  
let myName = "Aristea Kontogianni";
```

✘ ▶ Uncaught ReferenceError: Cannot access 'myName' before initialization

let

Οι μεταβλητές που ορίζονται με let/const έχουν Block Scope

- Block scope (w3schools): “Variables declared inside a { } block cannot be accessed from outside the block”

```
function foo(){
  if(true){
    var fruit1 = 'apple';      //exist in function scope
    const fruit2 = 'banana';  //exist in block scope
    let fruit3 = 'strawberry'; //exist in block scope
  }
  console.log(fruit1); //apple
  console.log(fruit2); //Uncaught ReferenceError: fruit2 is not defined at foo
  console.log(fruit3); //Uncaught ReferenceError: fruit3 is not defined at foo
}
```

Const

- Οι μεταβλητές που ορίζονται με τη λέξη – κλειδί const
- δεν μπορούν να δηλωθούν ξανά
- πρέπει να δηλώνονται πριν από τη χρήση
- έχουν Block Scope
- πρέπει αρχικοποιούνται όταν δηλώνονται
- Επίσης, δεν μπορούμε να εκχωρήσουμε εκ νέου τιμή σε const variable

} όπως με το Pet

Const

- Οι μεταβλητές που ορίζονται με τη λέξη – κλειδί `const` πρέπει αρχικοποιούνται όταν δηλώνονται

```
const PI;  
PI = 3.14;
```

✘ Uncaught SyntaxError: Missing initializer in const declaration

Const

- Δεν μπορούμε να εκχωρήσουμε εκ νέου τιμή σε const variable

```
const PI = 3.141592653589793;  
PI = 3.14;  
PI = PI + 10;
```

✘ ▶ Uncaught TypeError: Assignment to constant variable.

Const

- Χρησιμοποιώντας τη λέξη – κλειδί `const` δημιουργούμε μια `read-only` αναφορά σε μια τιμή
- Αυτό **δεν** σημαίνει ότι η τιμή που διατηρεί είναι **αμετάβλητη** - απλώς ότι **το όνομα της μεταβλητής δεν μπορεί να εκχωρηθεί εκ νέου**.
- Για παράδειγμα, στην περίπτωση που το περιεχόμενο μιας `const` μεταβλητής είναι ένα **αντικείμενο**, τα περιεχόμενά του (π.χ., οι ιδιότητές του) μπορούν να τροποποιηθούν.

Παράδειγμα

- Η JavaScript υποστηρίζει την δημιουργία objects, δηλαδή αντικειμένων
- Τα αντικείμενα είναι επίσης μεταβλητές. Αλλά τα αντικείμενα μπορούν να περιέχουν πολλές τιμές.
- ```
const person = {
 firstName: "Aristea",
 lastName: "Kontogianni",
 age: 19,
 eyeColor: "blue"
};
```

# Const

Έστω ότι έχουμε το παρακάτω array, τι περιμένουμε να συμβεί?

```
const cats = ["javie", "rosa", "gatoulis"];
cats[0] = "Javie";
cats.push("miaou");
console.log(cats);
```

# Const

```
// You can create a constant array:
const cats = ["javie", "rosa", "gatoulis"];

// You can change an element:
cats[0] = "Javie";

// You can add an element:
cats.push("miaou");

console.log(cats);
```

▶ (4) ['Javie', 'rosa', 'gatoulis', 'miaou']

---

# Const

Το παρακάτω θα δουλέψει?

```
const cats = ["javie", "rosa", "gatoulis"];
cats = ["Javie", "rosa", "gatoulis", "miaou"];
```

# Const

Δεν μπορούμε να εκχωρήσουμε εκ νέου τιμή στο cats

```
const cats = ["javie", "rosa", "gatoulis"];
cats = ["Javie", "rosa", "gatoulis", "miaou"];
```

✘ ▶ Uncaught TypeError: Assignment to constant variable.

# Ερώτηση!

- Ποιες από τις παρακάτω γραμμές κώδικα επιτρέπονται?

```
6 const x = 2;
7 const x = 3;
8 x = 3;
9 var y = 3;
10 let y = 3;
11
12 {
13 const x = 3;
14 x = 3;
15 let y = 3;
16 }
17
```

# Ερώτηση!

- Ποιες από τις παρακάτω γραμμές κώδικα επιτρέπονται?

```
6 const x = 2;
7 const x = 3;
8 x = 3;
9 var y = 3;
10 let y = 3;
11
12 {
13 const x = 3;
14 x = 3;
15 let y = 3;
16 }
17
```

```
<!DOCTYPE html>
<html>
<body>
<script>

const x = 2; // Allowed
//const x = 3; // Not allowed
//x = 3; // Not allowed
var y = 3; // allowed
//let y = 3; // Not allowed

{
 const x = 3; // Allowed
 //x = 3; // Not allowed
 let y = 3; // allowed
}

</script>
```

# JavaScript Identifiers

- JavaScript variables : ορίζονται χρησιμοποιώντας κάθε φορά μοναδικά ονόματα
- unique names : identifiers

Δημιουργία ονομάτων μεταβλητών:

- Αποδεκτοί χαρακτήρες: γράμματα, ψηφία, underscores, και dollar signs.
- Τα ονόματα των μεταβλητών μπορούν να ξεκινήσουν με ένα γράμμα ή με \$ και \_



# JavaScript Identifiers

- Δημιουργία ονομάτων μεταβλητών:
- Τα ονόματα είναι case sensitive
  - `var name, Name //different variables`
- Οι δεσμευμένες λέξεις δεν μπορούν να χρησιμοποιηθούν ως ονόματα μεταβλητών
- i.e. `for, int, static, true` etc

# Variable Scope

- Στην JavaScript οι μεταβλητές μπορεί να είναι
  - **Global Variables:** είναι προσβάσιμες από παντού στον κώδικα (ακόμα και από μέσα από τις συναρτήσεις) και δηλώνονται έξω από οποιαδήποτε συνάρτηση
  - **Local Variables :** Είναι ορατή μόνο μέσα στην συνάρτηση στην οποία ορίζεται

# Ερώτηση

- Τι θα προβληθεί στα στοιχεία με id=demo και id=exp ;

```
<p id="demo"></p>
<p id="exp"></p>

<script>
 var name = "Aristea";
 var surname = " Kontogianni";
 var z = name + surname;
 var x=4, y=3, total=x+y;

 document.getElementById("demo").innerHTML = z ;
 document.getElementById("exp").innerHTML = ""+total+"
 ";
</script>
```

# Απάντηση

- Τι θα προβληθεί στα στοιχεία με id=demo και id=exp ;

```
<p id="demo"></p>
<p id="exp"></p>

<script>
 var name = "Aristea";
 var surname = " Kontogianni";
 var z = name + surname;
 var x=4, y=3, total=x+y;

 document.getElementById("demo").innerHTML = z ;
 document.getElementById("exp").innerHTML = ""+total+"
 ";
</script>
```

Aristea Kontogianni

7

# Συμβολοσειρές (Strings)

- Οι συμβολοσειρές στη JavaScript, χρησιμοποιούνται για την αποθήκευση και την επεξεργασία κειμένου
- Μια συμβολοσειρά αποτελείται από έναν αριθμό χαρακτήρων μέσα σε εισαγωγικά (μονά/διπλά)
- `var name = "Aristea Kontogianni";`

# Συμβολοσειρές Μέθοδοι

charAt(position)	Returns the character at the specified position (in Number).
charCodeAt(position)	Returns a number indicating the Unicode value of the character at the given position (in Number).
concat([string,,])	Joins specified string literal values (specify multiple strings separated by comma) and returns a new string.
indexOf(SearchString, Position)	Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.
lastIndexOf(SearchString, Position)	Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.
localeCompare(string,position)	Compares two strings in the current locale.
match(RegExp)	Search a string for a match using specified regular expression. Returns a matching array.
replace(searchValue, replaceValue)	Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue.
search(RegExp)	Search for a match based on specified regular expression.

slice(startNumber, endNumber)	Extracts a section of a string based on specified starting and ending index and returns a new string.
split(separatorString, limitNumber)	Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.
substr(start, length)	Returns the characters in a string from specified starting position through the specified number of characters (length).
substring(start, end)	Returns the characters in a string between start and end indexes.
toLocaleLowerCase()	Converts a string to lower case according to current locale.
toLocaleUpperCase()	Converts a sting to upper case according to current locale.
toLowerCase()	Returns lower case string value.
toString()	Returns the value of String object.
toUpperCase()	Returns upper case string value.
valueOf()	Returns the primitive value of the specified string object.

# Strings είναι immutable

Οι συμβολοσειρές στη JavaScript είναι αμετάβλητες

- δεν μπορούν να τροποποιηθούν μόλις δημιουργηθούν.
- μπορείτε να δημιουργήσετε μια νέα συμβολοσειρά από μια υπάρχουσα συμβολοσειρά.

# Strings είναι immutable

```
var test = 'JavaScript';
test = test + 'Something';
console.log(test);
```

```
test[0] = 'X';
console.log(test);
console.log(test[0]);
```



# Objects

- Η JavaScript υποστηρίζει την δημιουργία objects, δηλαδή αντικειμένων
- Ένα αντικείμενο στη JavaScript είναι μια **συλλογή ονοματισμένων τιμών**
- Στα αντικείμενα λοιπόν έχουμε ζευγάρια key-value (name-value)
- Τα αντικείμενα ορίζονται με τις λέξεις κλειδιά var-let-const

# Objects

```
const cat = {
 name: "Javie",
 age: "5",
 color: "white",
 eyeColor: "blue"
};

console.log(cat["name"]);
console.log(cat.age);
```

Javie
5

# Objects

- Για να αποκτήσουμε πρόσβαση σε μια ιδιότητα ενός αντικειμένου, χρησιμοποιούμε:
- dot (.)
  - `objectName.propertyName`
- Array-like notation ( `[]` )
  - `objectName['propertyName']`
  - `objectName[expression]` // `x = " 'propertyName "`; `objectName[x]`
- Αν προσπαθήσουμε να διαβάσουμε μια ιδιότητα που δεν υπάρχει: `undefined`
- Μπορούμε να διαγράψουμε μια ιδιότητα μέσω του delete operator **//delete** `cat.name;`
- in operator: υπάρχει μια ιδιότητα σε ένα αντικείμενο?

# Objects

- Επιπλέον μπορούν να χρησιμοποιηθούν μεταβλητές ως ιδιότητες

```
var name="Aristea";
var surname = "Kontogianni";
var person = { name, surname };
console.log(person);
```

# Objects

- Επιπλέον μπορούν να χρησιμοποιηθούν μεταβλητές ως ιδιότητες

```
var name="Aristea";
var surname = " Kontogianni";
var person = { name, surname };
console.log(person);
```



▶ {name: 'Aristea', surname: ' Kontogianni'}

# Objects

- Τρόποι για να αποκτήσουμε πρόσβαση στις ιδιότητες ενός αντικειμένου :

```
var name="Aristea";
var surname = " Kontogianni";
var person = { name, surname };
console.log(person.name+" "+person["surname"]);
```

Aristea Kontogianni

# Objects

- Επίσης μπορούμε να προσθέσουμε εμφωλευμένο αντικείμενο μέσα σε ένα object
- Κάθε αντικείμενο στη JavaScript μπορεί να μετατραπεί σε συμβολοσειρά με τη συνάρτηση `JSON.stringify()`:

```
var name="Aristea";
var surname = " Kontogianni";
var person = { name, surname };
console.log(person.name+" "+person["surname"]);

var city ="Athens";
var country="Greece";
person.address={city,country};
console.log(person);
document.body.innerHTML = JSON.stringify(person);
```

```
{"name":"Aristea","surname":" Kontogianni","address":{"city":"Athens","country":"Greece"}}
```

# Objects

- JSON.parse()->string becomes a JavaScript object

```
var str= '{"name":"John", "age":30, "city":"New York"}';
console.log(typeof(str));
console.log(str);

const obj = JSON.parse(str);
console.log(typeof(obj));
console.log(obj);
```



# More on Objects

- `let user = {  
    name: "aristea",  
    surname: "Kontogianni",  
    age: "23"  
}; // "object literal" syntax`

Άλλος τρόπος δημιουργίας object : **new** keyword

```
let person = new Object();
person.name="John";
person.surname="Black";
person.age=30;
```

# Prototype inheritance

- Κάθε object στη JavaScript έχει μια ενσωματωμένη ιδιότητα, η οποία ονομάζεται **πρωτότυπο** (prototype).
- Το πρωτότυπο είναι από μόνο του ένα **αντικείμενο**, επομένως το **πρωτότυπο** θα έχει το δικό του πρωτότυπο, δημιουργώντας αυτό που ονομάζεται prototype chain.
- Η αλυσίδα τελειώνει όταν φτάσουμε σε ένα πρωτότυπο που έχει null για το δικό του πρωτότυπο.
- Τα Prototypes είναι ο μηχανισμός με τον οποίο τα JavaScript objects κληρονομούν χαρακτηριστικά το ένα από το άλλο

# Prototype inheritance

- Τα Prototypes είναι ο μηχανισμός με τον οποίο τα JavaScript objects κληρονομούν χαρακτηριστικά το ένα από το άλλο
- Όλα τα JavaScript objects κληρονομούν ιδιότητες και μεθόδους από ένα πρωτότυπο (Prototype), το οποίο είναι ένα object:  
Object.prototype

```
▼ {name: 'John', surname: 'Black', age: 30} ⓘ
 age: 30
 name: "John"
 surname: "Black"
 ► [[Prototype]]: Object
```

# Prototype inheritance

- Όταν προσπαθούμε να αποκτήσουμε πρόσβαση σε μια ιδιότητα ενός αντικειμένου: εάν η ιδιότητα δεν μπορεί να βρεθεί στο ίδιο το αντικείμενο, γίνεται αναζήτηση στο prototype για την ιδιότητα.
- Εάν η ιδιότητα εξακολουθεί να μην μπορεί να βρεθεί, τότε γίνεται αναζήτηση του πρωτοτύπου του πρωτοτύπου και ούτω καθεξής, μέχρι να φτάσουμε στο τέλος της αλυσίδας και να πάρουμε **undefined**

# Object create

- Η μέθοδος `Object.create()` δημιουργεί ένα νέο αντικείμενο, χρησιμοποιώντας ένα υπάρχον αντικείμενο ως πρωτότυπο του νεοδημιουργημένου αντικειμένου.

```

0
1 const person = {
2 name: 'Aristea',
3 age: 31
4 };
5
6 var me = Object.create(person);
7
8 me.surname = 'Kontogianni'; // "sur
9 me.email = 'test@gmail.com'; // "em
0
1 console.log(me);
2 console.log(me.name);
3 console.log(person.surname);
4
5
6

```

```

object_create.html:21
▼ {surname: 'Kontogianni', email: 'test@gmail.com'} ⓘ
 email: "test@gmail.com"
 surname: "Kontogianni"
 ▼ [[Prototype]]: Object
 age: 31
 name: "Aristea"
 ▼ [[Prototype]]: Object
 ▶ constructor: f Object()
 ▶ hasOwnProperty: f hasOwnProperty()
 ▶ isPrototypeOf: f isPrototypeOf()
 ▶ propertyIsEnumerable: f propertyIsEnumerable()
 ▶ toLocaleString: f toLocaleString()
 ▶ toString: f toString()
 ▶ valueOf: f valueOf()
 ▶ __defineGetter__: f __defineGetter__()
 ▶ __defineSetter__: f __defineSetter__()
 ▶ __lookupGetter__: f __lookupGetter__()
 ▶ __lookupSetter__: f __lookupSetter__()
 · __proto__: (...)
 ▶ get __proto__: f __proto__()
 ▶ set __proto__: f __proto__()
 Aristea object_create.html:22
 undefined object_create.html:23
>

```

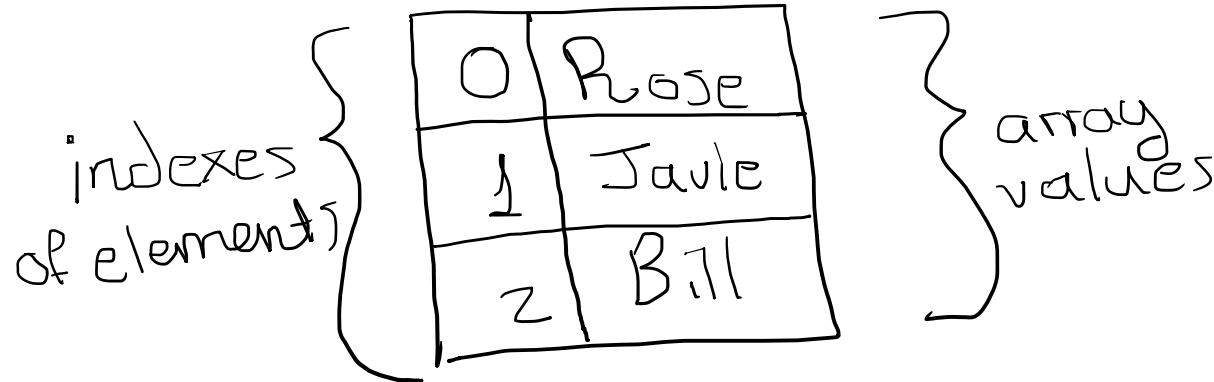
```

▶ {surname: 'Kontogianni', email: 'test@gmail.com'} object_create.html:21
Aristea object_create.html:22
undefined object_create.html:23
>

```

# Arrays

- Ένα array (πίνακας) αποτελεί μια μεταβλητή που περιέχει περισσότερες από μία τιμές κάθε φορά
- Στη μνήμη, ένα array αντιπροσωπεύει μια ομάδα τοποθεσιών μνήμης
- Κάθε μεμονωμένη τοποθεσία ονομάζεται στοιχείο (element)



# Arrays

- Cats -> Δημιουργούμε ένα array με τέσσερα στοιχεία, τα οποία ορίζονται κατά τη δήλωση του πίνακα

```
var cats = ["Ragdoll", "Bengal", "Ocicat", "Toyger"];
document.getElementById("demo").innerHTML = cats;
```

# Arrays

- Πόσα στοιχεία περιέχουν οι παρακάτω πίνακες;

```
<script>
var cats = ["Ragdoll", "Bengal", "Ocicat", "Toyger"];
var cats1 = ["Ragdoll",,,, "Toyger"];
```



# Arrays

- Cats -> Δημιουργούμε ένα array με **τέσσερα** στοιχεία, τα οποία **ορίζονται κατά τη δήλωση** του πίνακα
- Cats1-> Δημιουργεί ένα array με **5** στοιχεία, **3** από τα οποία δεν έχουν καθοριστεί ακόμη

```
var cats1 = ["Ragdoll", , , "Toyger"];
cats1[1]="Bengal";
document.body.innerHTML =cats1;
```

Ragdoll,Bengal,,Toyger

# typeof Array

- Οι πίνακες είναι στην πραγματικότητα ένας ειδικός τύπος αντικειμένων
- Έτσι, `typeof Array` επιστρέφει "object"
- Ωστόσο, τα `arrays` στη JavaScript είναι καλύτερο να περιγράφονται ως `arrays`

# Arrays' Elements

Ένα array μπορεί να περιέχει :

- μεταβλητές διαφορετικών τύπων
- αντικείμενα
- συναρτήσεις
- άλλα arrays

# Lets understand operators better...

$X=5$

==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

# Assignment Operators

Operator	This is the same as :	this
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x \% = y$	$x = x \% y$
^=	$x \wedge = y$	$x = x \wedge y$

## Interesting sources

<https://javascript.plainenglish.io/why-does-javascript-have-both-null-and-undefined-6a42fcca9301>

Check deepFreeze function: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Object/freeze#what\\_is\\_shallow\\_freeze](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze#what_is_shallow_freeze)

<https://medium.com/@ashfaqueahsan61/time-complexities-of-common-array-operations-in-javascript-c11a6a65a168>



*To be continued...*