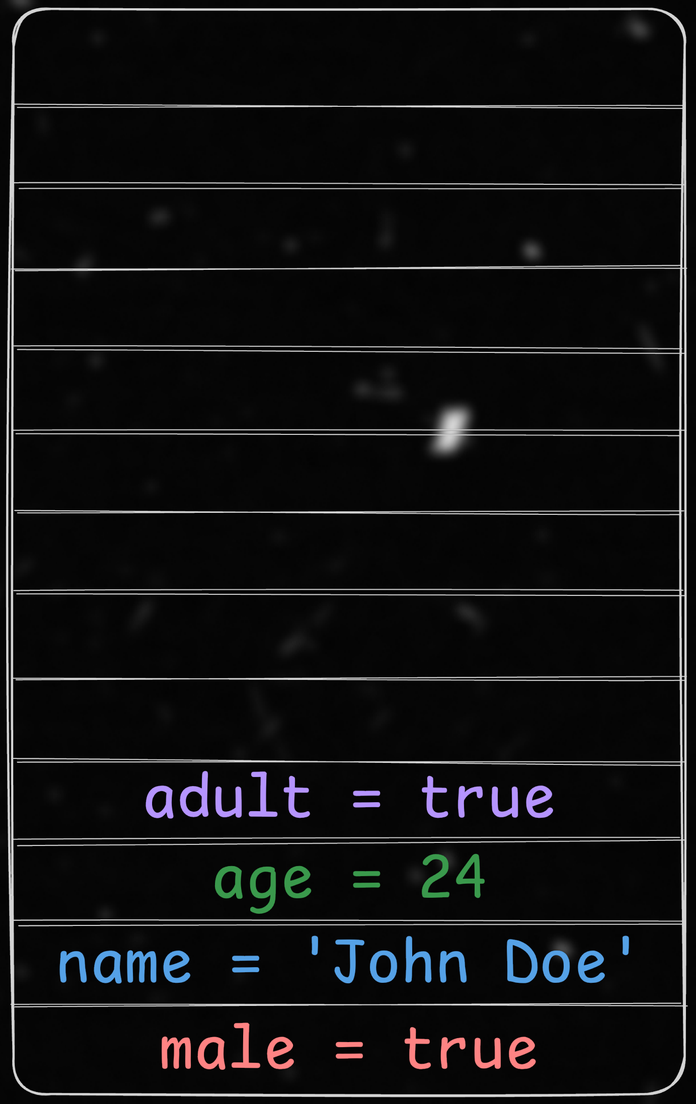
NodeJS

LABA

MEMORY LIFE CYCLE

Solvd
NodeJS LABA

STACK




```
const male = true;
const name = 'John Doe';
const age = 24;
const adult = true;
```

All the values are stored on the stack since they all contain primitive values

Solvd NodeJS LABA

STACK

* Primitive values and references
* Size is known at compile time
* Allocates a fixed amount of memory

HEAP

* Objects and Functions
* Size is known at runtime
* No limit per object

STACK

JS allocates memory for this object in the heap. The actual values are still primitives. Which is why they are stored in the stack

```
const person = {
  name: 'Jhon',
  age: 24,
}
```

STACK

Arrays are objects as well, which is why they are stored in the heap.

```
const hobbies = [
  "hiking",
  "reading"
];
```

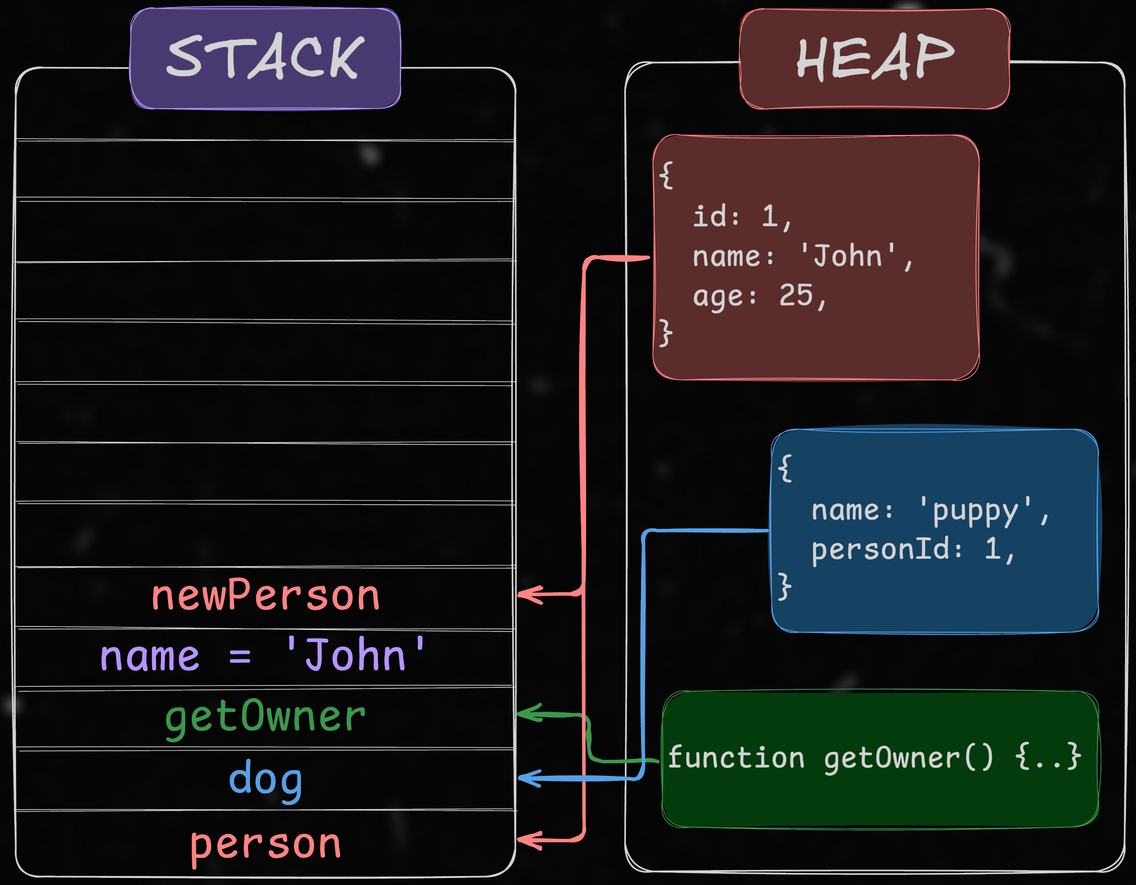
STACK

Primitive values are immutable, which means that instead of changing the original value, Javascript creates a new one.

```
// Allocates memory for a string
let name = 'Jhon';
// Allocates memory for a number
const age = 24;

// Allocates memory for a new string
name = 'John Doe';
// Allocates memory for a new string
const firstName = name.slice(0, 4)
```

STACK

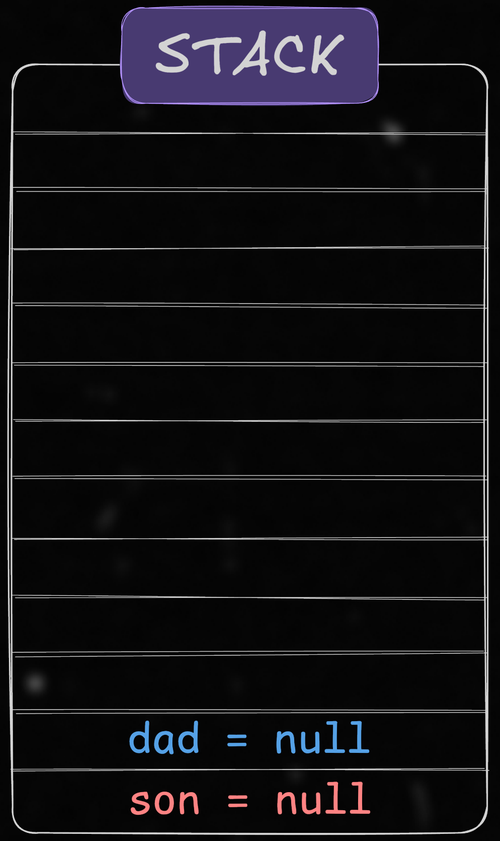
```
const person = {
  id: 1,
  name: 'John',
  age: 25,
}

const dog = {
  name: 'puppy',
  personId: 1,
}

function getOwner(dog, persons) {
  return persons.find(
    (person) => person.id === dog.person
  )
}

const name = 'John'
const newPerson = person;
```

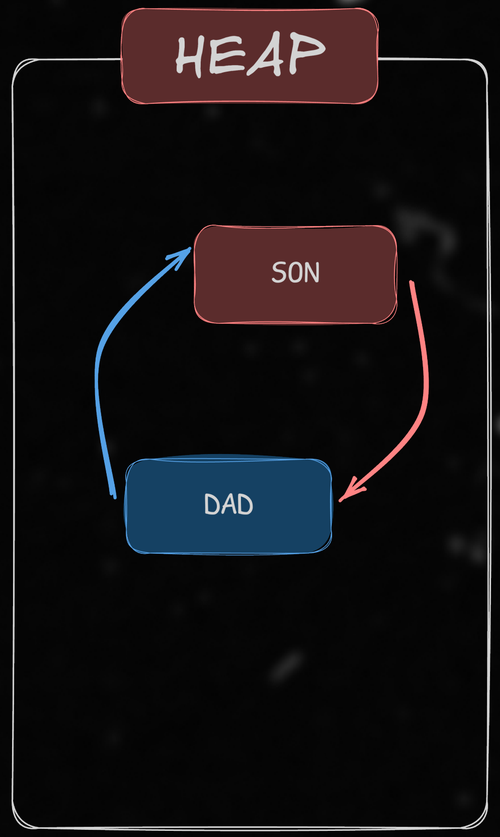


Solvd
NodeJS LABA

STACK

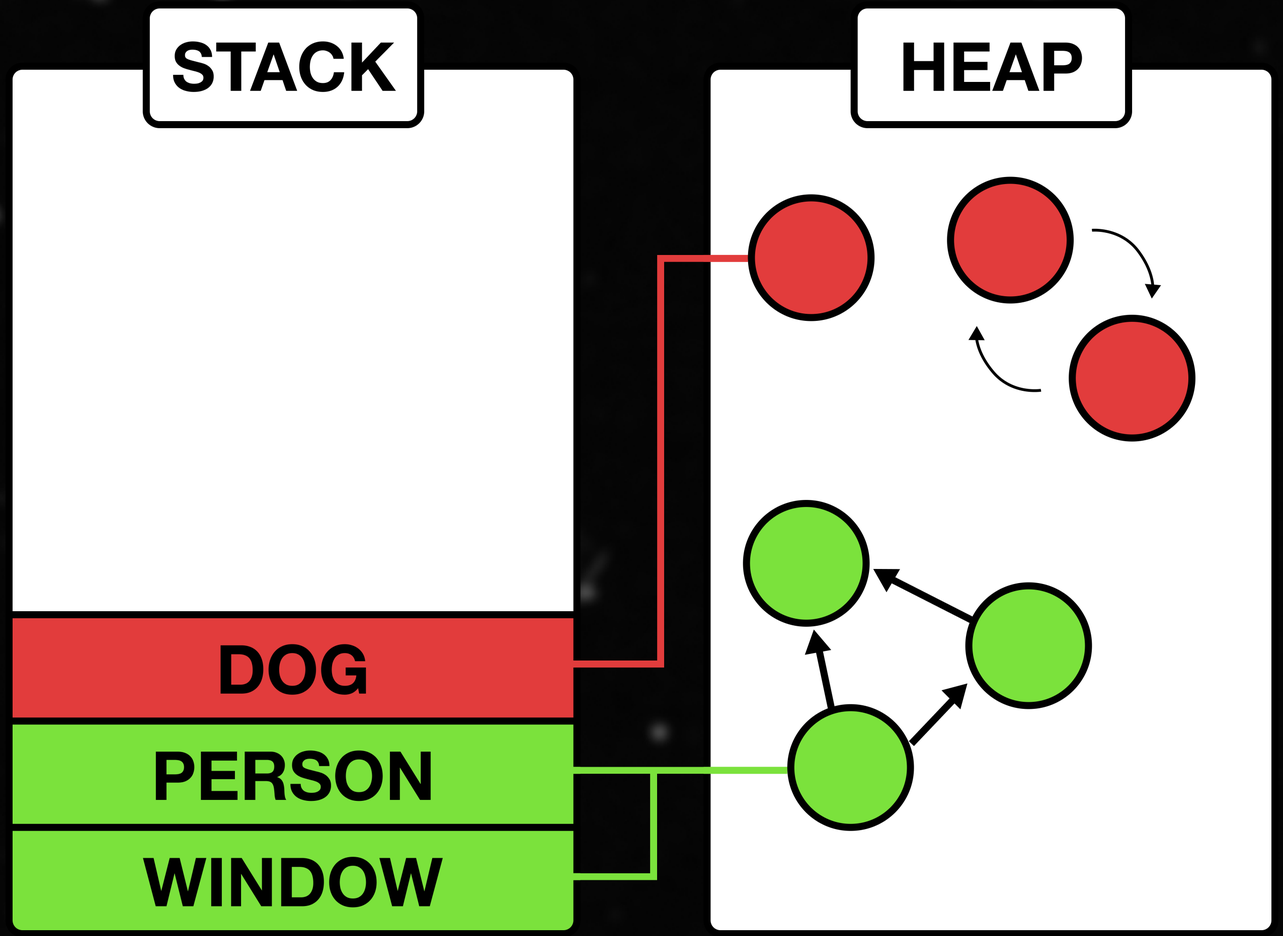
```
let son = {
  name: 'Johnson'
}

let dad = {
  name: 'John'
}

son.dad = dad
dad.son = son

son = null
dad = null
```





Solvd
NodeJS LABA

STACK



Solvd
NodeJS LABA

GARBAGE COLLECTION

```
user = getUser();

var secondUser = getUser();

function getUser() {
  return 'user';
}
```

Solvd NodeJS LABA

GARBAGE COLLECTION

```
const object = {}

const intervalId = setInterval(function() {
  // everything used in here can't be collected
  // until the interval is cleared
  doSomething(object)
}, 2000);
```

Solvd NodeJS LABA

GARBAGE COLLECTION

Clearing intervals is specially important in SPAs. Even when navigating away from the page where this interval is needed, it will still run in the background.

Old browsers have problems collecting unused event listeners. Its a good idea to remove them after using.

```
clearInterval(intervalId);
```

GARBAGE COLLECTION

```
const element = document.getElementById('button');
const onClick = () => alert('hi');

element.addEventListener('click', onClick);

element.removeEventListener('click', onClick);
element.parentNode.removeElement(element);
```

Solvd NodeJS LABA

COLLECTIONS

```
const map = new Map();

map.set('1', 'str1');
map.set(1, 'num1');
map.set(true, 'bool1');

// remember the regular Object? it would convert keys to string
// Map keeps the type, so these two are different:
alert(map.get(1)); // 'num1'
alert(map.get('1')); // 'str1'

alert(map.size); // 3
```

COLLECTIONS

```
let john = { name: 'John' };

// for every user, let's store their visits count
let visitsCountMap = new Map();

// john is the key for the map
visitsCountMap.set(john, 123);

alert(visitsCountMap.get(john)); // 123
```

COLLECTIONS

```
let john = { name: 'John' };
let ben = { name: 'Ben' };

let visitsCountObj = {};
visitsCountObj[ben] = 234;
visitsCountObj[john] = 123;

// That's what got written
alert(visitsCountObj["[object Object]"]); // 123
```

COLLECTIONS

Every map.set call returns the map itself, so we can “chain” the calls:

```
map.set('1', 'str1')
   .set(1, 'num1')
   .set(true, 'bool1');
```

COLLECTIONS

```
const recipeMap = new Map([
  ['cucumber', 500],
  ['tomatoes', 350],
  ['onion',    50],
]);

// iterate over keys (vegetables)
for (let vegetable of recipeMap.keys()) {
  alert(vegetable); // cucumber, tomatoes, onion
}

for (let amount of recipeMap.values()) {
  alert(amount); // 500, 350, 50
}

for (let entry of recipeMap) {
  alert(entry); // cucumber,500 (and so on)
}
```

COLLECTIONS

```
// runs the function for each (key, value) pair
recipeMap.forEach((value, key, map) => {
  alert(`${key}: ${value}`); // cucumber: 500 etc
});
```

Solvd NodeJS LABA

COLLECTIONS

```
// array of [key, value] pairs
const map = new Map([
  ['1',  'str1'],
  [1,    'num1'],
  [true, 'bool1']
]);

alert(map.get('1')); // str1
```

COLLECTIONS

```
const obj = {
  name: "John",
  age: 30
};

let map = new Map(Object.entries(obj));

alert(map.get('name')); // John
```

Solvd NodeJS LABA

COLLECTIONS

```
const prices = Object.fromEntries([
  ['banana', 1],
  ['orange', 2],
  ['meat',   4]
]);

// now prices = { banana: 1, orange: 2, meat: 4 }
alert(prices.orange); // 2
```

COLLECTIONS

```
const map = new Map();
map.set('banana', 1);
map.set('orange', 2);
map.set('meat',   4);

// make a plain object
let obj = Object.fromEntries(map.entries());

// Done!
// obj = { banana: 1, orange: 2, meat: 4 }

alert(obj.orange); // 2

obj = Object.fromEntries(map); // omit .entries()
```

Solvd
NodeJS LABA

COLLECTIONS

```
let set = new Set();

let john = { name: 'John' };
let pete = { name: 'Pete' };
let mary = { name: 'Mary' };

// visits, some users come multiple times
set.add(john);
set.add(pete);
set.add(mary);
set.add(john);
set.add(mary);

// set keeps only unique values
alert(set.size); // 3

for (let user of set) {
  alert(user.name); // John (then Pete and Mary)
}
```

COLLECTIONS

```
const set = new Set(["oranges", "apples", "bananas"]);

for (let value of set) alert(value);

// the same with forEach
set.forEach((value, valueAgain, set) => {
  alert(value);
});
```

Solvd NodeJS LABA

COLLECTIONS

The object previously referenced by John is stored inside the array, therefore it won’t be garbage-collected. We can get it as array[0];

```
const john = { name: 'John' };
const array = [ john ];

john = null; // overwrite the reference
```

COLLECTIONS

John is stored inside the map, we can get it by using map.keys()

```
const john = { name: 'John' };

const map = new Map();
map.set(john, "...");

john = null; // overwrite the reference
```

COLLECTIONS

```
const weakMap = new WeakMap();

const obj = {};

weakMap.set(obj, "ok"); // works fine (object key)

// can't use a string as the key
weakMap.set("test", "whoops");
// Error, because "test" is not an object
```

Solvd
NodeJS LABA

COLLECTIONS

```
const john = { name: 'John' };

const weakMap = new WeakMap();
weakMap.set(john, "...");

john = null; // overwrite the reference
// john is removed from memory
```

COLLECTIONS

```
let visitsCountMap = new Map();

function countUser(user) {
  let count = visitsCountMap.get(user) || 0;
  visitsCountMap.set(user, count + 1);
}
```

COLLECTIONS

```
let john = { name: 'John' };
countUser(john); // count his visits

// later john leaves us
john = null
```

COLLECTIONS

```
let visitsCountMap = new WeakMap( );

function countUser(user) {
  let count = visitsCountMap.get(user) || 0;
  visitsCountMap.set(user, count + 1);
}
```

Solvd NodeJS LABA

COLLECTIONS

```
let cache = new Map();

// calculate and remember the result
function process(obj) {
  if (!cache.has(obj)) {
    // calculations of the result for obj
    let result = obj;

    cache.set(obj, result);
    return result;
  }

  return cache.get(obj);
}

// now we use process() in another file
let obj = { /* let's say we have an object */ };

let result1 = process(obj); // calculated

// later, from another place on the code...
let result2 = process(obj);
// remembered result taken from cache

// ...later when the object is not needed anymore
obj = null;

// size is 1 (Ouch, the object is still in cache)
alert(cache.size);
```

Solvd
NodeJS LABA

COLLECTIONS

```
let cache = new WeakMap();

// calculate and remember the result
function process(obj) {
  if (!cache.has(obj)) {
    // calculations of the result for obj
    let result = obj;

    cache.set(obj, result);
    return result;
  }

  return cache.get(obj);
}

let obj = { /* let's say we have an object */ };

let result1 = process(obj);
let result2 = process(obj);

// ...later when the object is not needed anymore
obj = null;

// can't get cache.size as it's a WeakMap,
// but its 0 or will soon be 0
// when obj gets garbage collected.
```

Solvd
NodeJS LABA

COLLECTIONS

```
let visitedSet = new WeakSet();

let john = { name: 'John' };
let pete = { name: 'Pete' };
let mary = { name: 'Mary' };

visitedSet.add(john); // John visited us
visitedSet.add(pete); // Then Pete
visitedSet.add(john); // John again

// visitedSet has 2 users now

// check if John visited?
alert(visitedSet.has(john)) // true

// check if Mary visited?
alert(visitedSet.has(mary)) // false

john = null;

// visitedSet will be cleaned automatically
```

COLLECTIONS

```
let student = {
  name: 'John',
  age: 30,
  isAdmin: false,
  courses: ['html', 'css', 'js'],
  spouse: null
};

let json = JSON.stringify(student);

alert(typeof json); // we've got a string!

alert(json);
/* JSON-encoded object:
{
  "name": "John",
  "age": 30,
  "isAdmin": false,
  "courses": ["html", "css", "js"],
  "spouse": null
}
*/
```

COLLECTIONS

```
alert(JSON.stringify(1)) // 1

// a string in JSON is still a string, but double-quoted
alert(JSON.stringify('test')) // "test"

alert(JSON.stringify(true)) // true

alert(JSON.stringify([1, 2, 3])) // [1,2,3]
```

Solvd NodeJS LABA

COLLECTIONS

```
let user = {
  sayHi() { // ignored
    alert("Hello");
  },
  [Symbol("id")]: 123, // ignored
  something: undefined // ignored
};

alert(JSON.stringify(user)) // {} (empty object)
```

Solvd
NodeJS LABA

COLLECTIONS

```
let meetup = {
  title: "Conference",
  room: {
    number: 23,
    participants: ["john", "ann"]
  }
};

alert(JSON.stringify(meetup));
/* The whole structure is stringified:
{
  "title": "Conference",
  "room": {"number":23,"participants":["john","ann"]}
}
*/
```

Solvd NodeJS LABA

COLLECTIONS

Here, the conversion fails, because of a circular reference: room.occupiedBy references meetup, and meetup.place references room

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: ["john", "ann"]
};

meetup.place = room; // meetup references room
room.occupiedBy = meetup; // room references meetup

JSON.stringify(meetup);
// Error: Converting circular structure to JSON
```

COLLECTIONS

```
let json = JSON.stringify(value[, replacer, space])

let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup references room
};

room.occupedBy = meetup; // room references meetup

alert(JSON.stringify(meetup, ['title', 'participants']));
// {"title":"Conference","participants":[{},{}]}
```

COLLECTIONS

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup references room
};

room.occupiedBy = meetup;

alert(JSON.stringify(meetup, [
  'title',
  'participants',
  'place',
  'name',
  'number',
]));

/*
{
  "title":"Conference",
  "participants": [{"name":"John"},{"name":"Alice"}],
  "place":{"number":23}
}
*/
```

Solvd NodeJS LABA

COLLECTIONS

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup references room
};

room.occupiedBy = meetup;

alert(JSON.stringify(meetup, function replacer(key, value) {
  alert(`${key}: ${value}`);
  return (key === "occupiedBy") ? undefined : value;
}));

/* key value pairs that come to replacer:
:                    [object Object]
title:               Conference
participants:        [object Object],[object Object]
0:                   [object Object]
name:                John
1:                   [object Object]
name:                Alice
place:               [object Object]
number:              23
occupiedBy:          [object Object]
*/
```

Solvd
NodeJS LABA

COLLECTIONS

```
let user = {
  name: "John",
  age: 25,
  roles: {
    isAdmin: false,
    isEditor: true
  }
};

alert(JSON.stringify(user, null, 2));
/* two-space indents:
{
  "name": "John",
  "age": 25,
  "roles": {
    "isAdmin": false,
    "isEditor": true
  }
}
*/
```

Solvd
NodeJS LABA

COLLECTIONS

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  date: new Date(Date.UTC(2017, 0, 1)),
  room
};

alert(JSON.stringify(meetup));
/*
{
  "title":"Conference",
  "date":"2017-01-01T00:00:00.000Z",
  "room":{"number":23}
}
*/
```

COLLECTIONS

```
let room = {
  number: 23,
  toJSON() {
    return this.number;
  }
};

let meetup = {
  title: "Conference",
  room
};

alert(JSON.stringify(room)); // 23

alert(JSON.stringify(meetup));
/*
{
  "title":"Conference",
  "room":23
}
*/
```

Solvd
NodeJS LABA

COLLECTIONS

```
let value = JSON.parse(str[, reviver]);

// stringified array
let numbers = "[0, 1, 2, 3]";

numbers = JSON.parse(numbers);

alert(numbers[1]); // 1
```

COLLECTIONS

```
let userData = '{"name":"John","age":35,"isAdmin":false}';

let user = JSON.parse(userData);

alert(user.age); // 35
```

COLLECTIONS

```
let json = `{
  name: "John",                  // mistake: property name without quotes
  "surname": 'Smith'             // mistake: single quotes in value
  'isAdmin': false               // mistake: single quotes in key
  "friends": [0,1,2,3]           // this is fine!
}`;
```

Solvd
NodeJS LABA