The image features decorative palm fronds in the top-left and top-right corners, framing the content. The fronds are green with detailed leaf patterns.

DESTRUCTURING  
NESTED OBJECT  
DOT NOTATION  
BRACKET NOTATION  
OPTIONAL CHAINING

# DESTRUCTURING IN JAVASCRIPT

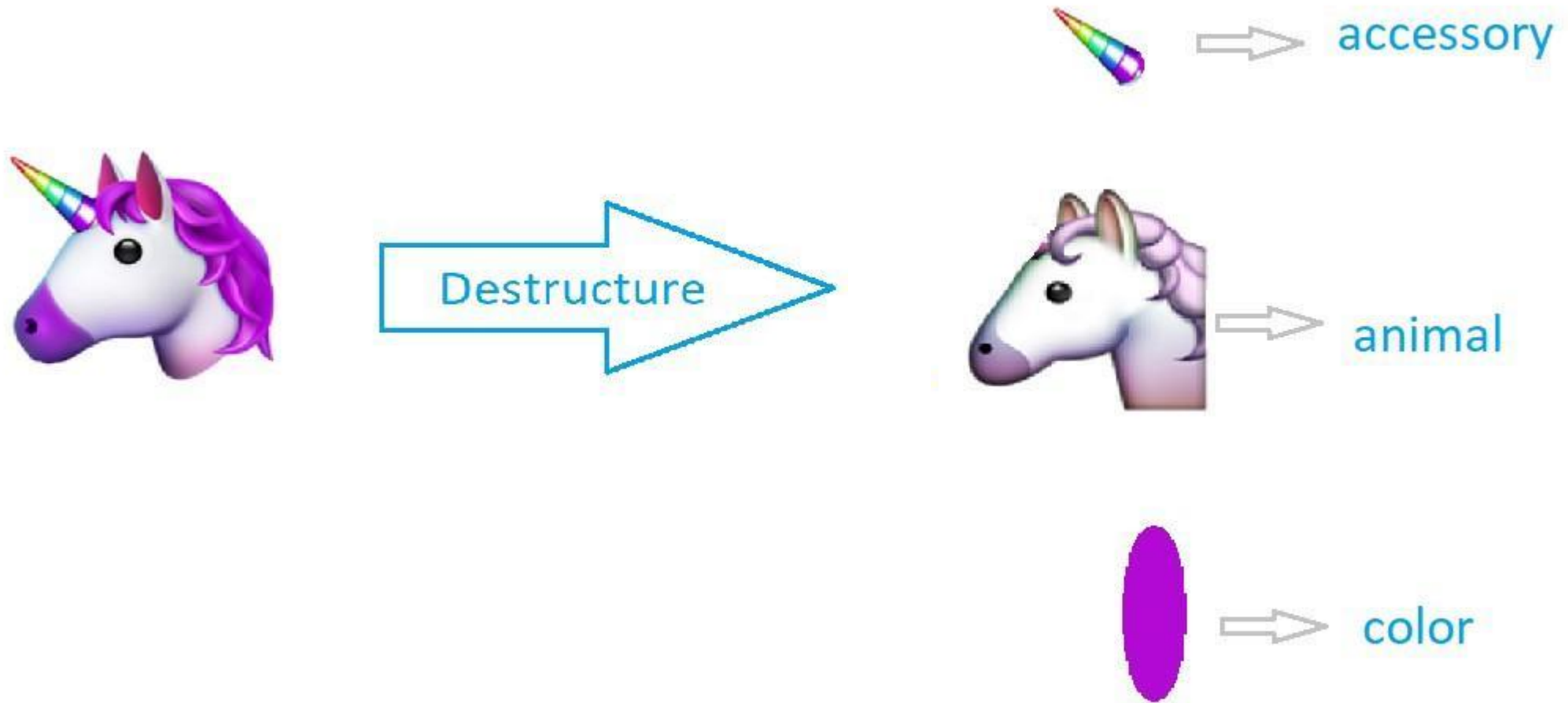




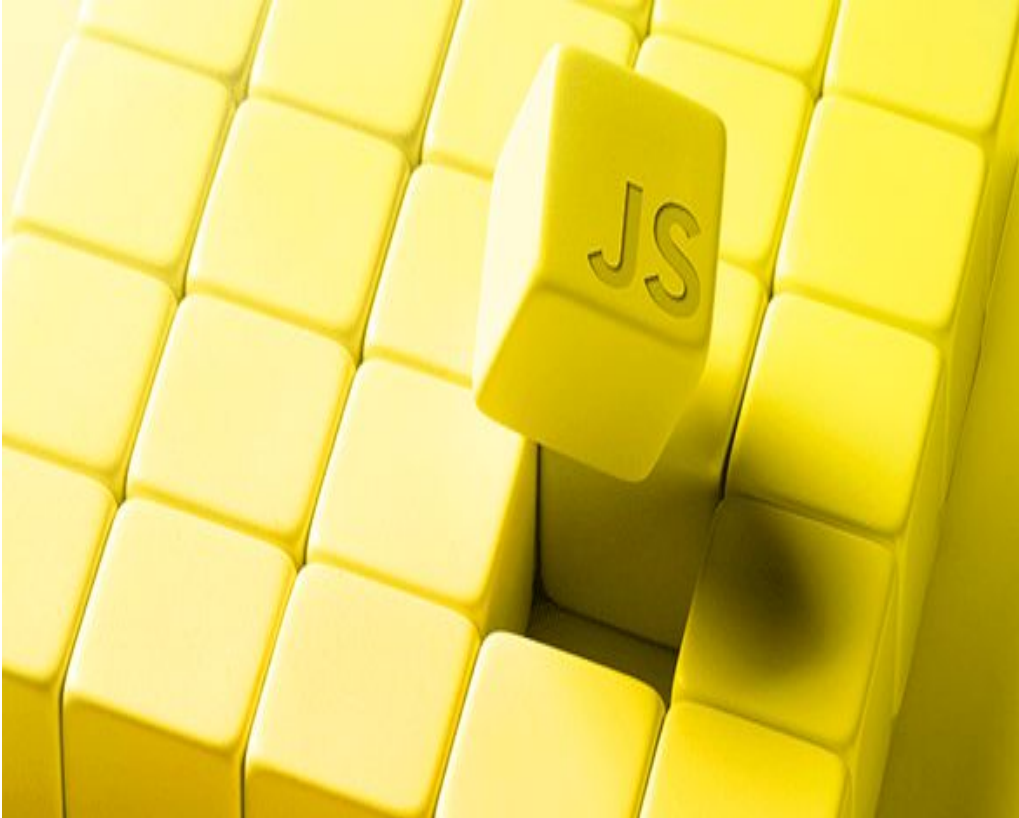
# What is destructuring?



# What is destructuring?



# Necessity of destructuring



- A technique for unpacking objects, arrays, and assigning them to variables
- Helps to deal with functions that have a lot of parameters, default values
- Is convenient to write, easy to maintain and friendly to read
- Saves from writing repetitive code





# OBJECT DESTRUCTURING



**JS**

# OBJECT DESTRUCTURING

**1**

## PROPERTY TO VARIABLE

```
const { prop } = object;
```

**2**

## MULTIPLE PROPERTIES

```
const { prop1, prop2, ..., propN }  
    = object;
```

**3**

## DEFAULT VALUE

```
const { prop = 'Default' } = object;
```

**4**

## ALIAS

```
const { prop: myProp } = object;
```

**5**

## DEEP PROPERTY

```
const { prop: { deepProp } } = object;
```

**6**

## DYNAMIC PROPERTY NAME


```
const { [propName]: myProp } = object;
```





JS

## OBJECT DESTRUCTURING



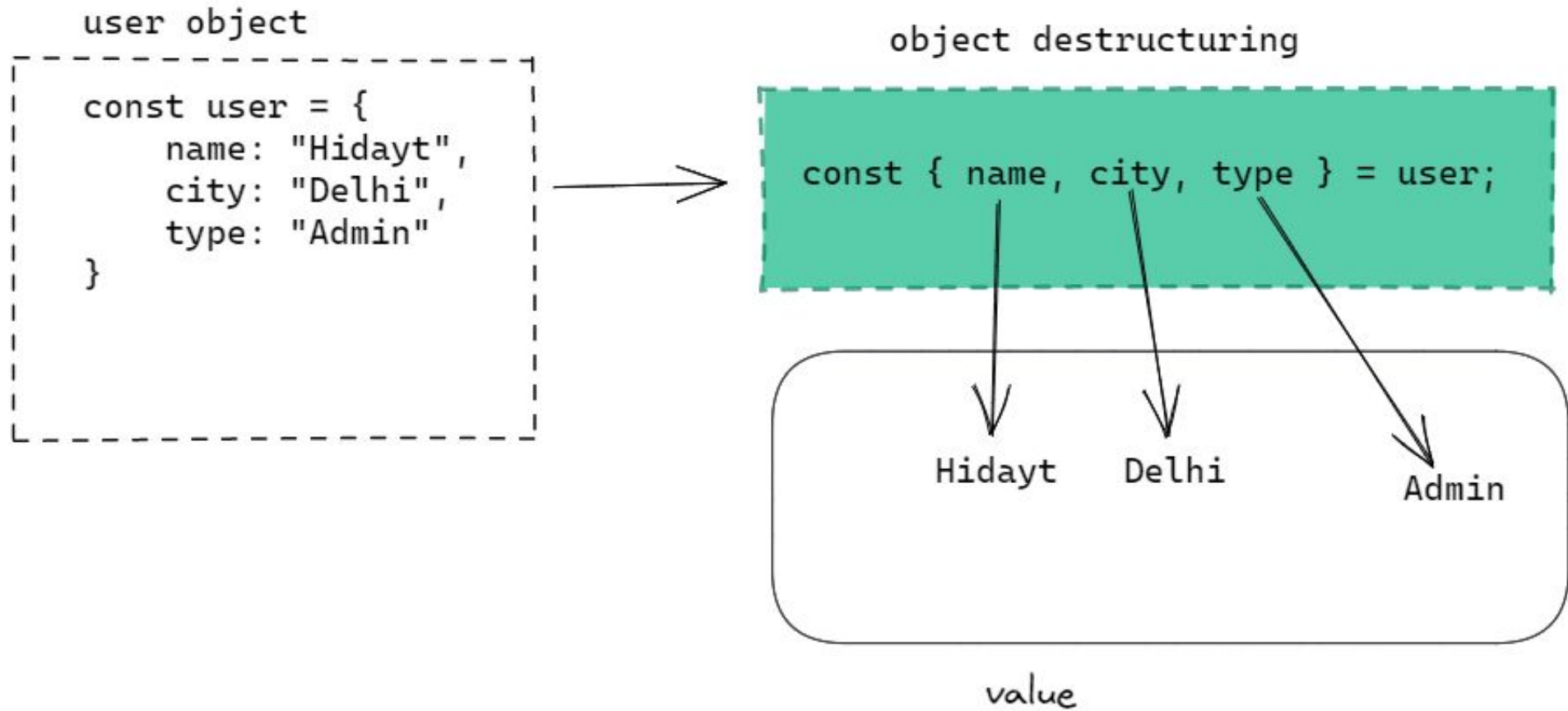
The image shows a screenshot of a web browser's developer console. At the top, there are three tabs: 'Inspector', 'Console' (which is selected and highlighted with a blue bar), and 'Debugger'. Below the tabs, there is a 'Filter Output' section with a trash can icon and a funnel icon. The main area of the console displays the following JavaScript code:

```
>> const user = {  
    'name' : 'Rowan Atkinson',  
    'popularity' : 'Mr. Bean'  
}  
  
const {name} = user;
```





# JS OBJECT DESTRUCTURING



# JS OBJECT DESTRUCTURING



```
const myObject = {  
  student: 'Mike',  
  teacher: 'Susan'  
};
```

```
const { student: pupil, teacher: prof } = myObject;
```

```
console.log(pupil, prof);  
> Mike Susan
```

```
const { student, teacher } = myObject;  
console.log(student, teacher)  
> Mike Susan
```



# JS OBJECT DESTRUCTURING

```
const user = {  
  id: 42,  
  user_name: 'dangtrunganh',  
  year: 1985,  
  is_verified: true  
};  
  
const {user_name: u, is_verified: iv, salary: s} = user;  
  
console.log(s) // undefined
```



# JS OBJECT DESTRUCTURING

★ Use destructuring to retrieve values from an object

```
const employee = {  
  id: 007,  
  name: 'James',  
  dept: 'Spy'  
}
```

```
const id = employee.id;  
const name = employee.name;
```

```
const { id, name } = employee;
```





# JS OBJECT DESTRUCTURING

★ Use destructuring to retrieve values from a nested object

```
const employee = {  
  id: 007,  
  name: 'James',  
  dept: {  
    id: 'D001',  
    name: 'Spy',  
    address: {  
      street: '30 Wellington Square',  
      city: 'Chelsea'  
    }  
  }  
}
```

```
const address = employee.dept.address;
```

It works, and the output is,

```
{  
  "street": "30 Wellington Square",  
  "city": "Chelsea"  
}
```



# JS OBJECT DESTRUCTURING

★ Use destructuring to retrieve values from a nested object

```
const employee = {  
  id: 007,  
  name: 'James',  
  dept: {  
    id: 'D001',  
    name: 'Spy',  
    address: {  
      street: '30 Wellington Square',  
      city: 'Chelsea'  
    }  
  }  
}
```

```
const street = employee.dept.address.street;
```

It works, and the output is,

```
30 Wellington Square
```



# JS OBJECT DESTRUCTURING

★ Define a new variable with object destructuring

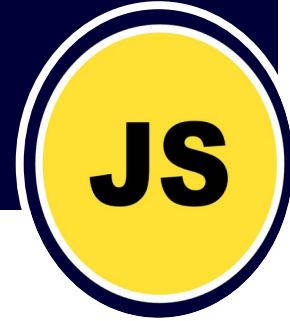
```
const employee = {  
  id: 007,  
  name: 'James',  
  dept: 'Spy'  
}
```

```
const age = employee.age ? employee.age : 25;
```

```
const { name, age=25 } = employee;  
console.log(age);
```



# **Destructuring Array**





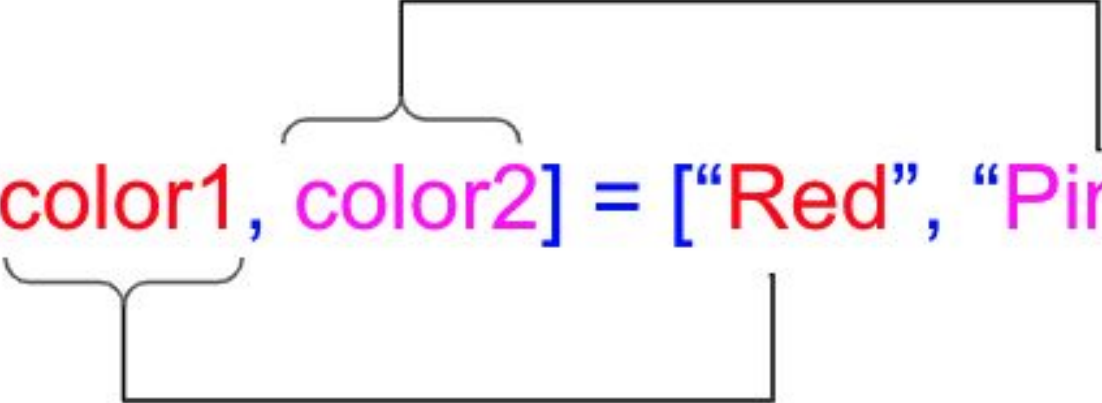
`var [`  `,`  `]` `=`  `;`



```
const [ ❤️, 🥑, 🏃 ] = ['love', 'avocado', 'runner']
```



Var [color1, color2] = ["Red", "Pink", "Blue", "Black"]



# Destructuring Array

JS

```
1 var numbers = [1, 2, 3];  
2 var num1 = numbers[0]  
3 var num2 = numbers[1]  
4 var num3 = numbers[2]  
5 console.log(num1)  
6 console.log(num2)  
7 console.log(num3)
```

CONSOLE ×

1  
2  
3

```
1 let numbers = [1, 2, 3];  
2 let [num1, num2, num3] = numbers;  
3 console.log(num1)  
4 console.log(num2)  
5 console.log(num3)
```

CONSOLE ×

1  
2  
3

```
1 let num1, num2, num3;  
2 [num1, num2, num3] = [1, 2, 3];  
3 console.log(num1)  
4 console.log(num2)  
5 console.log(num3)
```

CONSOLE ×

1  
2  
3

```
1 let [num1, num2, num3] = [1, 2, 3];  
2 console.log(num1)  
3 console.log(num2)  
4 console.log(num3)
```

CONSOLE ×

1  
2  
3



# Destructuring Array

JS

Assign variables with a default value:

```
1 let num1, num2, num3;  
2 [num1=0, num2=5, num3=7] = [1, 2];  
3 console.log(num1)  
4 console.log(num2)  
5 console.log(num3)
```

CONSOLE ×

1  
2  
7

Swapping Values:

```
1 let num1 = 1;  
2 let num2 = 2;  
3  
4 [num1, num2] = [num2, num1]; // swap  
5 console.log(num1)  
6 console.log(num2)  
7
```

CONSOLE ×

2  
1

## Skipping Items in an Array:

```
1 let [num1,, num3] = [1, 2, 3];  
2 console.log(num1)  
3 console.log(num3)
```

CONSOLE ×

1

3



## Destructuring with Functions:

```
1 function numbers(){  
2   | return [1, 2, 3, 4]  
3 }  
4 let [num1, num2] = numbers();  
5 console.log(num1);  
6 console.log(num2);  
7 |
```

CONSOLE ×

1  
2



Rest(...) operator:

```
1 let [num1, ...num2] = [1, 2, 3, 4, 5, 6];  
2 console.log(num1)  
3 console.log(num2)  
4
```

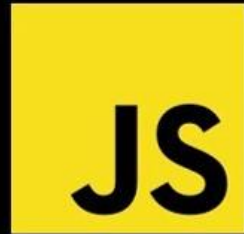
CONSOLE ×

1

(5) [2, 3, 4, 5, 6]







# Optional Chaining (?.)





# Optional Chaining (.?)

Using .

```
profiles.user.phone.number  
// TypeError
```

Using ?.

```
profiles?.user?.phone?.number  
// undefined
```

```
let profiles = {  
  user: {  
    name: 'supi',  
    email: 'supi@gmail.com'  
  }  
}
```





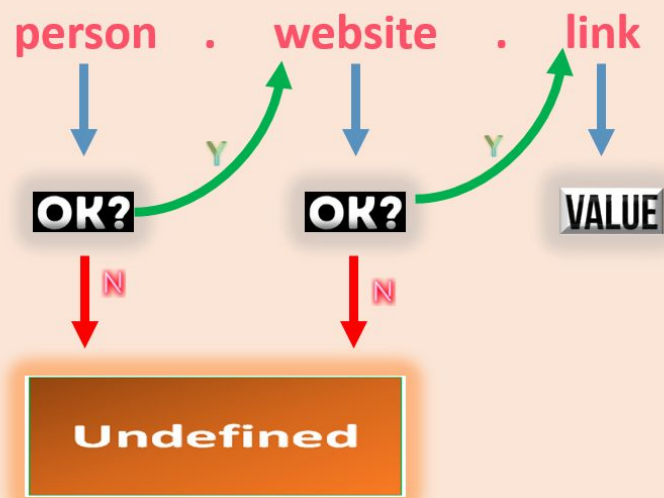
27

28

```
29 console.log(person.website.link); Cannot read property 'link' of null
```

30

31



```
>> let person = {  
  name: "Shakib Al Hasan",  
  wikipedia: {  
    nickName: "Faysal",  
    link: "https://en.wikipedia.org/wiki/Shakib_Al_Hasan"  
  }  
};  
undefined
```



```
1      var user = { name: 'Joe' }  
2  
3      // does not throw error  
4      var zip = user?.address?.zip  
5  
6      console.log(zip) // undefined 👍  
7
```





```
const person = {  
  name: 'xyz',  
  age: 25,  
  country: 'Bangladesh',  
  details: {  
    job: true,  
    dob: 'xyz',  
    isMarried: false  
  }  
}  
console.log(person.description?.isEating);  
// undefined
```



# Dot Notation VS Bracket Notation

● VS []





# Dot Notation VS Bracket Notation

- Dot notation is faster to write and clearer to read, easy to read.
- Square bracket notation allows access to properties containing special characters and selection of properties using variables.
- You can use variables with bracket notation, but not with dot notation. This is especially useful for situations when you want to access a property but don't know the name of the property ahead of time.



● vs []


```
const variable = 'happy';
const emotion = {
  happy: '😊',
  sad: '😭'
};

// Dot : Access Property
emotion.happy; // '😊'

// Bracket : Access Property With Variable
emotion[variable]; // '😊'
```



Common mistake you can make while accessing object property:

```
const variable = 'name';  
const obj = {  
  name: 'value'  
};  
// Bracket Notation  
obj[variable]; //  'value'  
// Dot Notation  
obj.variable; // undefined
```



# You have to use square bracket notation when- ● vs []

1. The property name is number.

```
var ob = {  
  1: 'One',  
  7 : 'Seven'  
}  
ob.7 // SyntaxError  
ob[7] // "Seven"
```

2. The property name has special character.

```
var ob = {  
  'This is one': 1,  
  'This is seven': 7,  
}  
ob.'This is one' // SyntaxError  
ob['This is one'] // 1
```

3. The property name is assigned to a variable and you want to access the property value by this variable.

```
var ob = {  
  'One': 1,  
  'Seven': 7,  
}  
  
var _Seven = 'Seven';  
ob._Seven // undefined  
ob[_Seven] // 7
```



## Cleaner code with dot notation

```
1  ✓ const people = {  
2  ✓    Neil: {  
3      firstName: "Neil"  
4    },  
5  ✓    bob: {  
6      firstName: "Bob"  
7    },  
8  ✓    stan: {  
9      firstName: "Stan"  
10   }  
11  };  
12  
13  // Dot Notation  
14  console.log(people.Neil.firstName);      // => Neil  
15  
16  // Bracket Notation  
17  console.log(people["Neil"]["firstName"]); // => Neil
```



## Dot Notation's Limitation

Accessing their propriety using dot notation:

```
obj.123;      // ✗ SyntaxError
obj.123name;   // ✗ SyntaxError
obj.name123;   // ✓ 'does not start with digit'
obj.$name;     // ✓ '$ sign'
obj.name-123;  // ✗ SyntaxError
obj.'name-123'; // ✗ SyntaxError
obj.NAME;      // ✓ 'upper case'
obj.name;      // ✓ 'lower case'
```

But none of this is a problem for the Bracket Notation:

```
obj['123'];    // ✓ 'digit'
obj['123name']; // ✓ 'start with digit'
obj['name123']; // ✓ 'does not start with digit'
obj['$name'];   // ✓ '$ sign'
obj['name-123']; // ✓ 'does not start with digit'
obj['NAME'];    // ✓ 'upper case'
obj['name'];    // ✓ 'lower case'
```

```
const obj = {
  123: 'digit',
  123name: 'start with digit',
  name123: 'does not start with digit',
  $name: '$ sign',
  name-123: 'hyphen',
  NAME: 'upper case',
  name: 'lower case'
};
```





**LET'S  
START  
NEXT  
TOPIC**

