DESTRUCTURING

NESTED OBJECT

DOT NOTATION

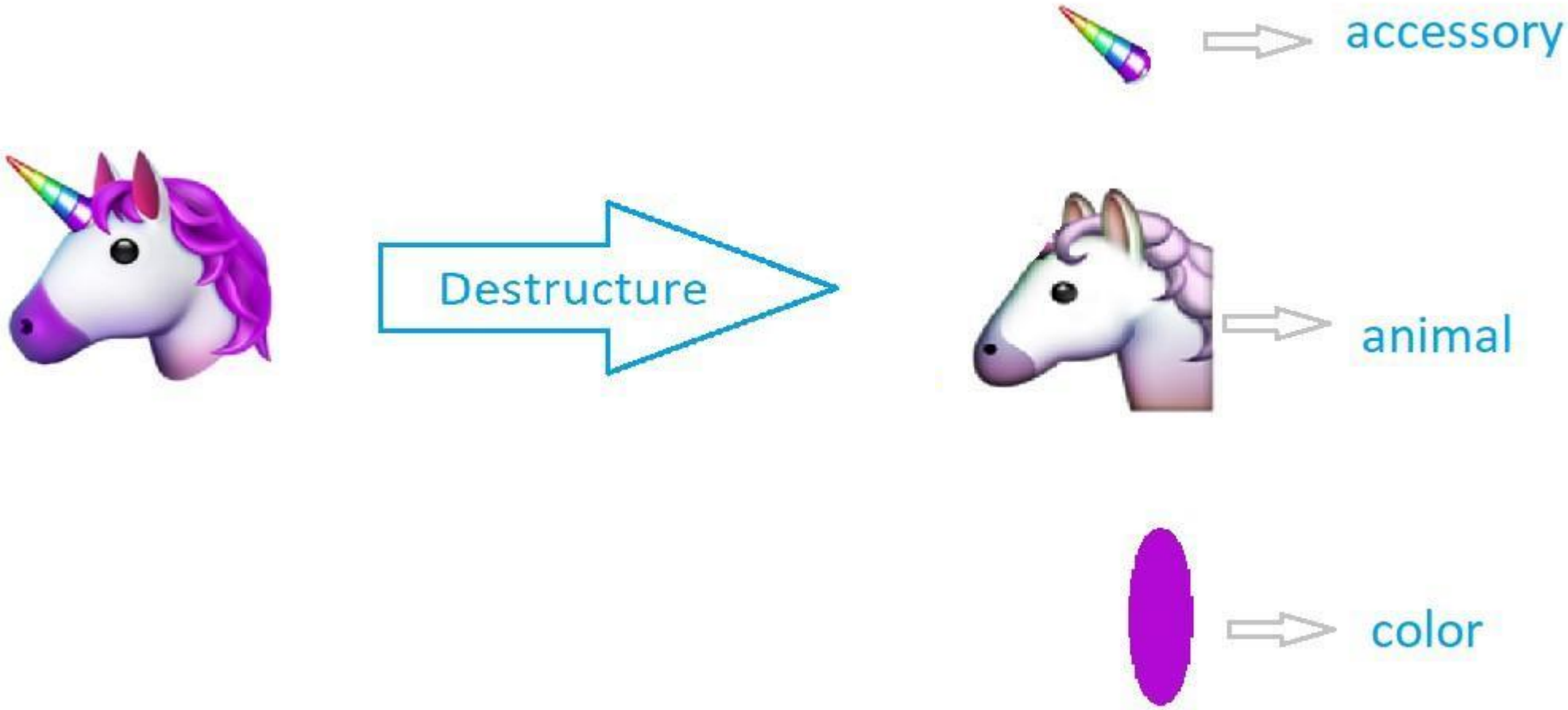
BRACKET NOTATION

OPTIONAL CHAINING

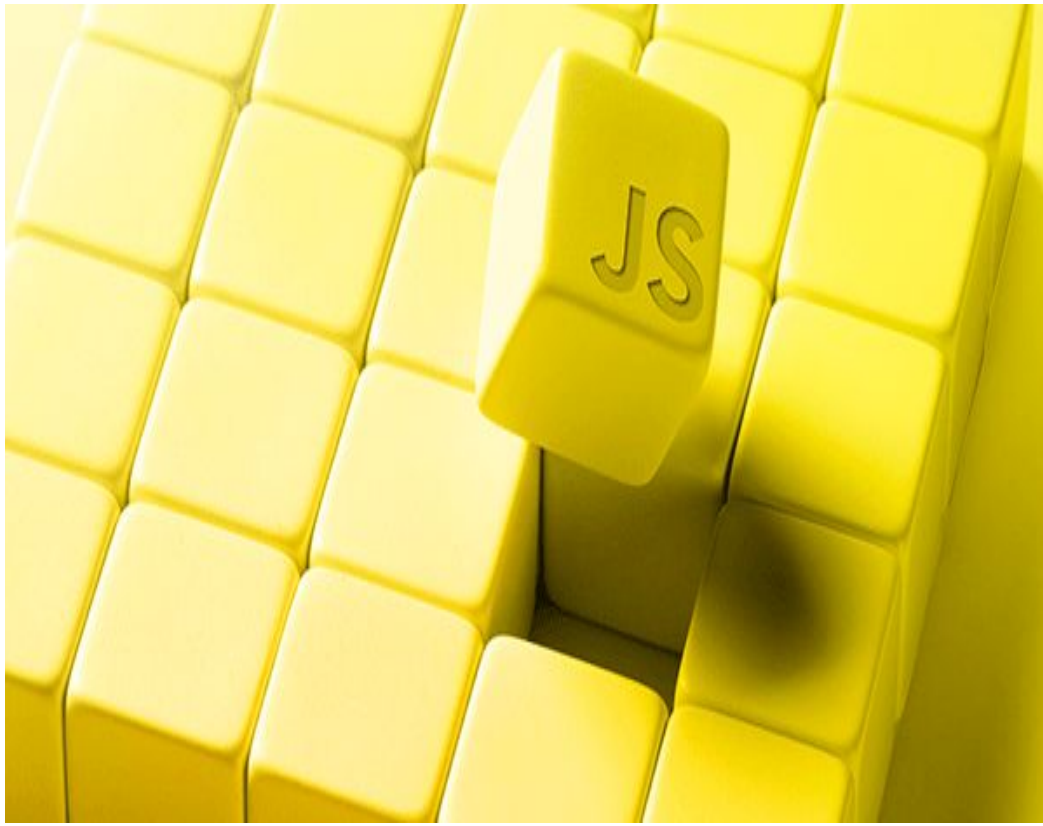
DESTRUCTURING IN JAVASCRIPT

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

ref: tmedium.com/swlh

JS

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

Inspector | Console | Debugger

Filter Output

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

   const {name} = user;
```

ref: tapasadhikary

JS OBJECT DESTRUCTURING

user object

```
const user = {
    name: "Hidayt",
    city: "Delhi",
    type: "Admin"
}
```

object destructuring

```
const { name, city, type } = user;
```

Hidayt Delhi Admin

value

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;
```

ref: greenroots.info

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"
}
```

ref: greenroots.info

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
```

ref: greenroots.info

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);
```

ref: greenroots.info

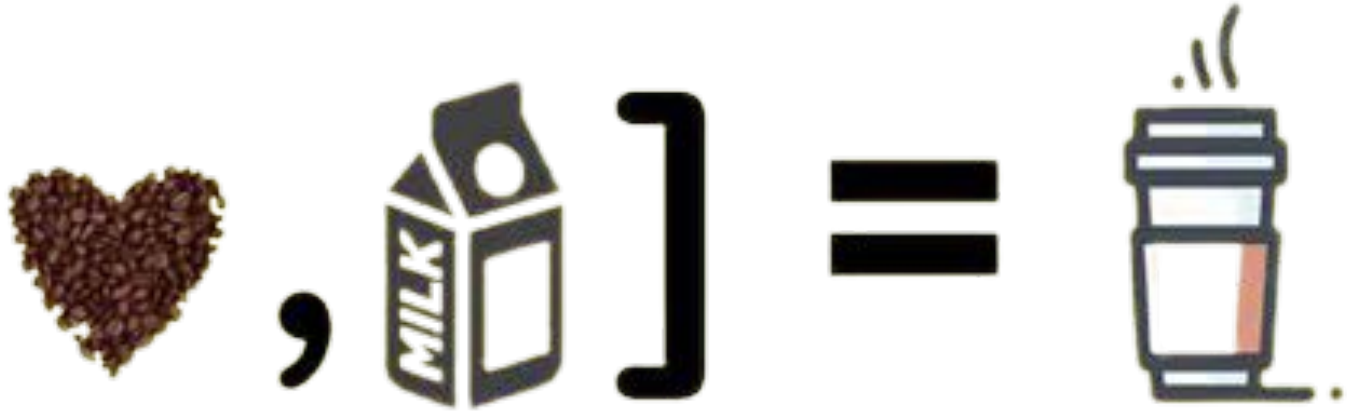
Destructuring Array

JS

Destructuring Array

JS

var [,] =



Destructuring Array

JS

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

Destructuring Array

JS

Var [color1, color2] = [“Red”, “Pink”, “Blue”, “Black”]

Destructuring Array

JS

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

CONSOLE

```
1
2
3
```

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

CONSOLE

```
1
2
3
```

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

CONSOLE

```
1
2
3
```

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

CONSOLE

```
1
2
3
```

ref: tmedium.com/swlh

Destructuring Array

JS

Assign variables with a default value:

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

CONSOLE

```
1
2
7
```

Swapping Values:

```
let num1 = 1;
let num2 = 2;

[num1, num2] = [num2, num1]; // swap
console.log(num1)
console.log(num2)
```

CONSOLE

```
2
1
```

ref: tmedium.com/swlh

Destructuring Array

JS

Skipping Items in an Array:

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

CONSOLE

```
1
3
```

ref: tmedium.com/swlh

Destructuring with Functions:

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

CONSOLE

```
1
2
```

ref: tmedium.com/swlh

Destructuring Array

JS

Rest(...) operator:

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

CONSOLE

```
1
(5) [2, 3, 4, 5, 6]
```

ref: tmedium.com/swlh

JS

Optional Chaining (?.)

Optional Chaining (.?)

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

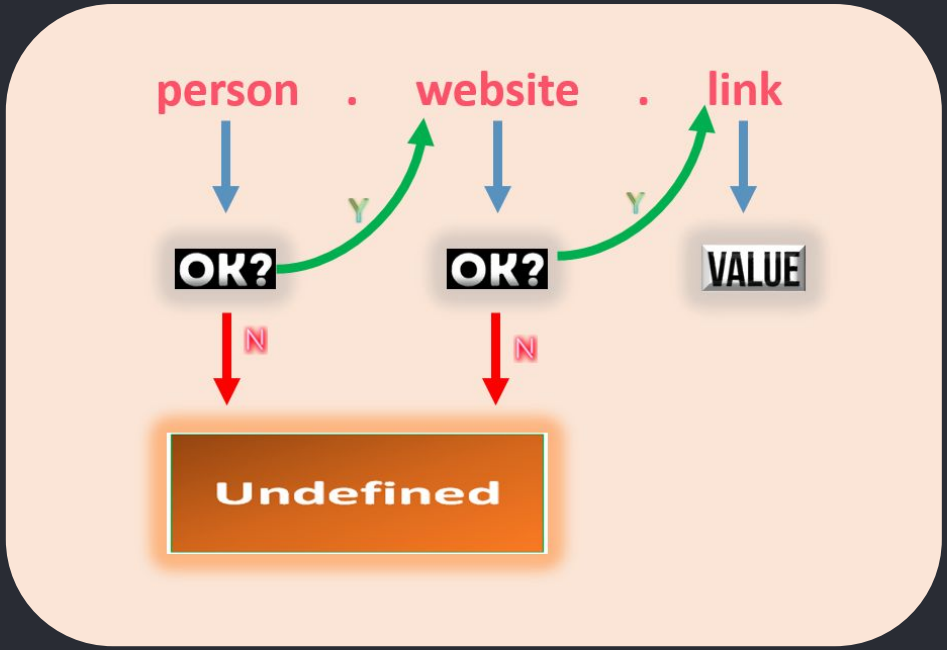
Using .

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

Using ?.

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

```
27
28
29  console.log(person.website.link);  Cannot read property 'link' of null
30
31
```



Inspector | Console | Debugger | Network | { }

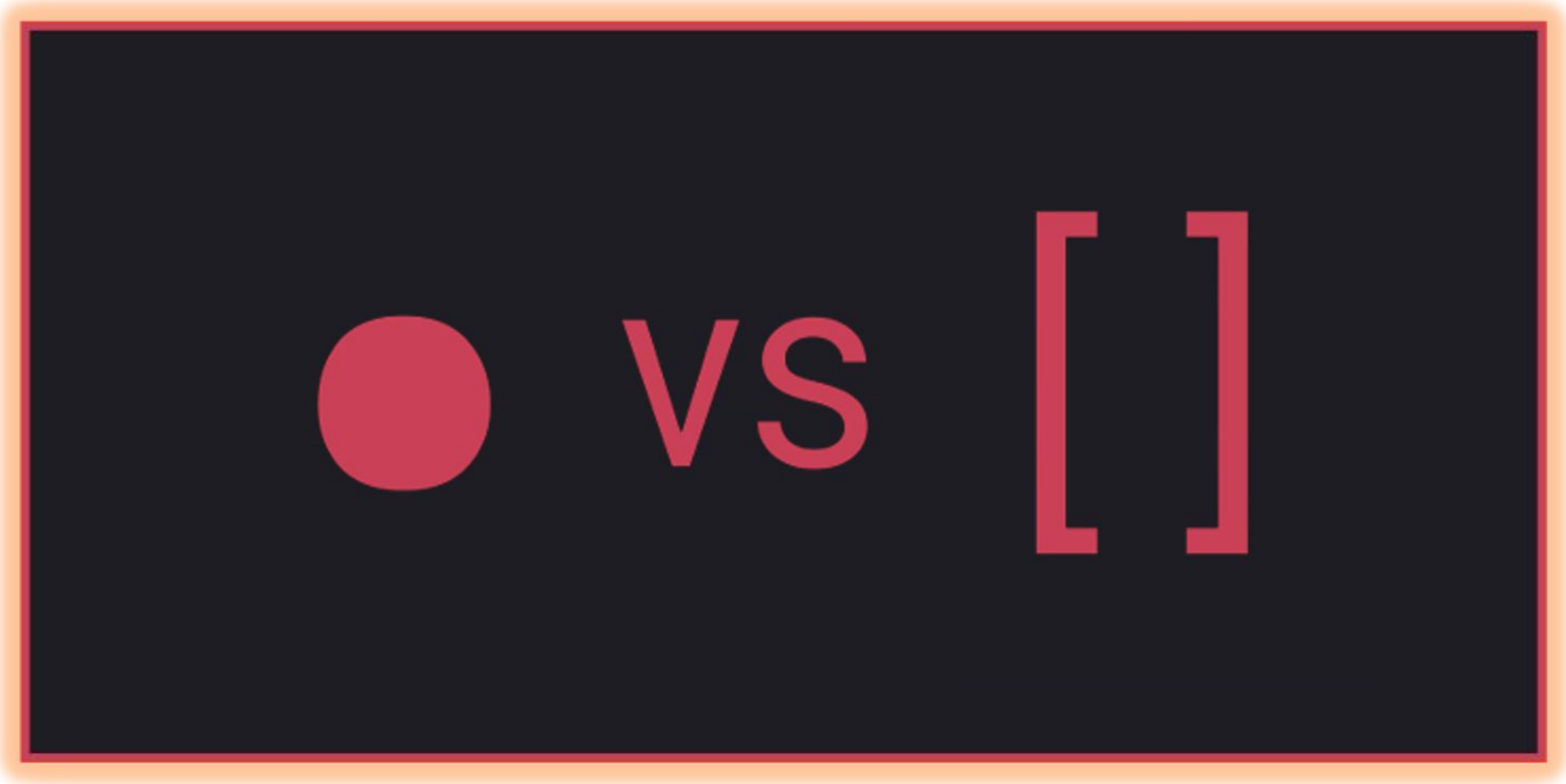
Filter Output | Errors

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

```
var user = { name: 'Joe' }

// does not throw error
var zip = user?.address?.zip

console.log(zip) // undefined 👍

```

```
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



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 [ ]

ref: stackoverflow
ref: javascript.plainenglish.io

. 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

```javascript
const people = {
    Neil: {
        firstName: "Neil"
    },
    bob: {
        firstName: "Bob"
    },
    stan: {
        firstName: "Stan"
    }
};

// Dot Notation
console.log(people.Neil.firstName);          // => Neil

// Bracket Notation
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