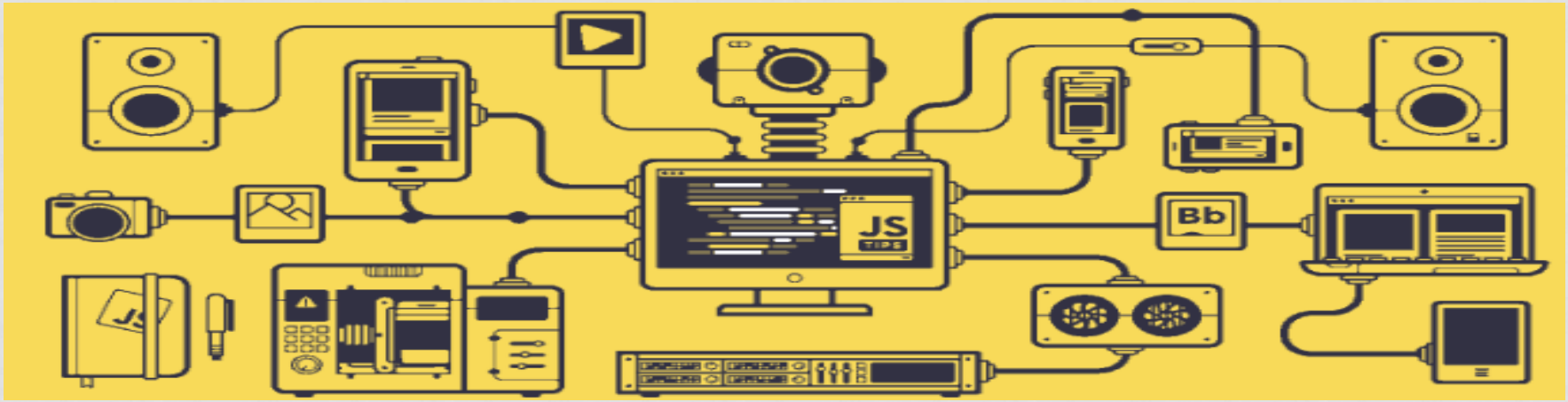


CLASSES/OBJECTS

TYPES IN JAVASCRIPT

PROTOTYPICAL

JavaScript is a prototype-based language. Every object in JavaScript has a hidden internal property called 'prototype'

PROTOTYPICAL

Attributes and functions can be removed from your classes.

Dynamically

NO PACKAGES

Java has packages

that can be used to group logic or features together.

In Javascript we can simulate that with classes (namespaces)

SPECIFICATION

The language specification ECMAScript 2015, often referred to as ES6, introduced classes to the JavaScript language

PURPOSE:
CREATE CUSTOM OBJECTS

Classes are just like functions. Both create objects.

NOT LIKE
JAVA, C++...

JavaScript's classes are not like classes in Java, C# . . . any other object-oriented language.

MORE **CONVENIENT**

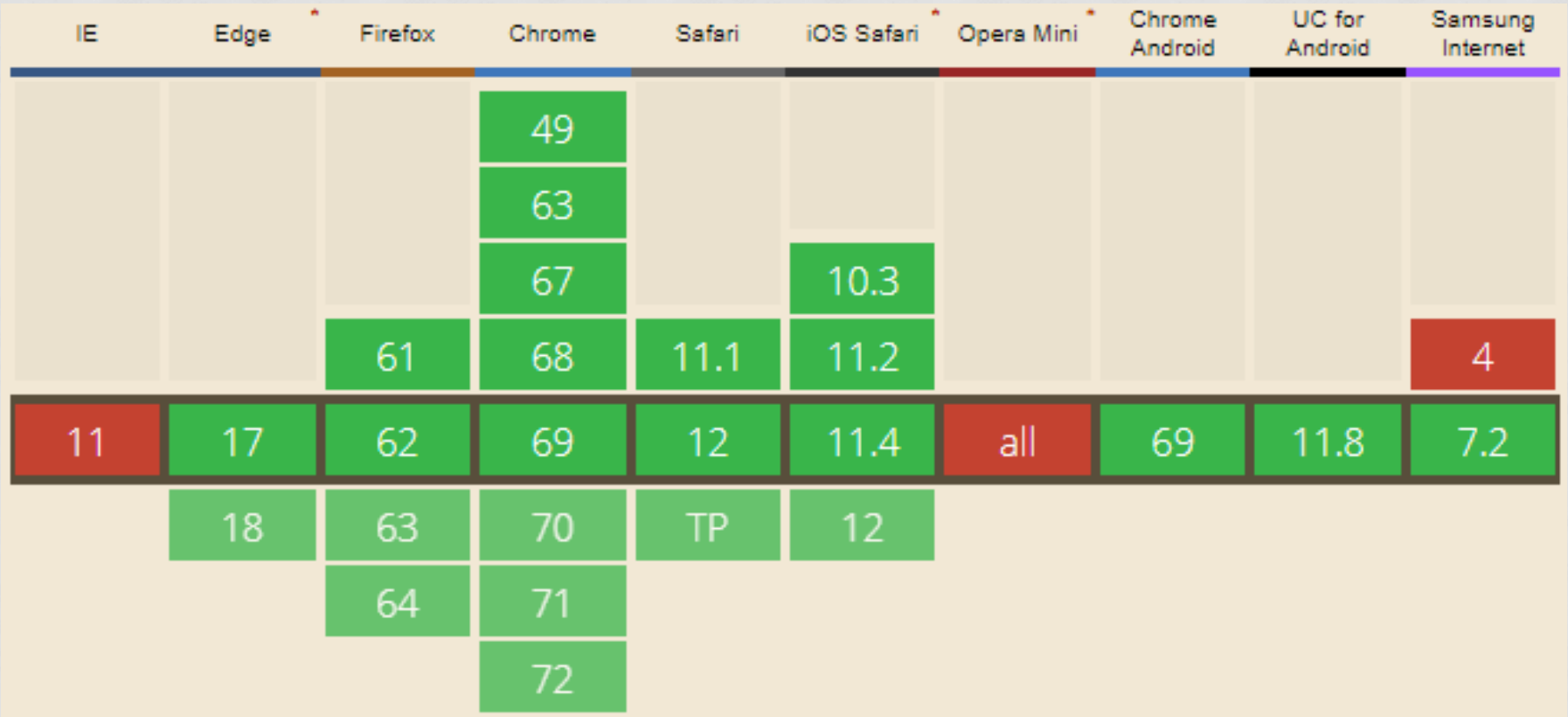
They mainly provide more convenient syntax to create old-school constructor functions.

THE CLASS KEYWORD

The class keyword is actually a invoking a function.

Or, in other words, another way to create a class object

CAN I USE? [YES]
(NOT WITH IE – ANY VERSION)



TYPES

JavaScript has 2 types

Reference types (Date, Array & String)

Primitive Types (Boolean, Numeric, Null, Undefined)

FUNCTION AND CLASS
SAME, BUT DIFFERENT

```
function Note(id, content, owner) {
    this.id = id;
    this.content = content;
    this.owner = owner;
}
```

```
class Note {
    constructor(id, content, owner) {
        this.id = id;
        this.content = content;
        this.owner = owner;
    }
}
```

CLASS EXPRESSIONS (JUST LIKE OBJECT LITERAL)

```
const Note = class {
    constructor(id, content, owner) {
        this.id = id;
        this.content = content;
        this.owner = owner;
    }
};
```

DECLARING **EXAMPLE**

```
1  class Rectangle {
2      constructor(height, width) {
3          this.height = height;
4          this.width = width;
5      }
6  }
```

DECLARING **EXAMPLE**

```
1  class Polygon {
2    constructor() {
3      this.name = "Polygon";
4    }
5  }
6
7  var poly1 = new Polygon();
8
9  console.log(poly1.name);
10 // expected output: "Polygon"
```

CLASS
BODY

```
1  class Rectangle {
2    constructor(height, width) {
3      this.height = height;
4      this.width = width;
5    }
6  }
```

CONSTRUCTOR EXAMPLE
INVOKING SUPER()

```
1  class Square extends Polygon {
2    constructor(length) {
3      // Here, it calls the parent class' constructor with lengths
4      // provided for the Polygon's width and height
5      super(length, length);
6      // Note: In derived classes, super() must be called before you
7      // can use 'this'. Leaving this out will cause a reference error.
8      this.name = 'Square';
9    }
10
11   get area() {
12     return this.height * this.width;
13   }
14
15   set area(value) {
16     this.area = value;
17   }
18 }
```

ONLY ONE CONSTRUCTOR

There can be only one method with the name "constructor" in a class.

Having more than one occurrence of a “constructor” in a class will throw a SyntaxError

CONSTRUCTOR
OPTIONAL

No constructor needed. If you don't have anything to indicate - skip it.

A default will be generated.

SUBCLASSES AND THE CONSTRUCTORS

If you use a constructor in your derived class.

You have to call super in the constructor.

METHOD DEFINITIONS

```
class Food {

    constructor (name, protein, carbs, fat) {
        this.name = name;
        this.protein = protein;
        this.carbs = carbs;
        this.fat = fat;
    }

    toString () {
        return `${this.name} | ${this.protein}
    }

    print () {
      console.log( this.toString() );
    }
}
```

STATIC METHODS

```
1  class ClassWithStaticMethod {
2    static staticMethod() {
3      return 'static method has been called.';
4    }
5  }
6
7  console.log(ClassWithStaticMethod.staticMethod());
8  // expected output: "static method has been called."
9
10
```

DERIVED **CLASSES**

```
class FatFreeFood extends Food {

    constructor (name, protein, carbs) {
        super(name, protein, carbs, 0);
    }

    print () {
        super.print();
        console.log(`Would you look at that -- ${this.name} has no fat!`);
    }

}
```

ACCESSOR **PROPERTIES**

```
1  class Square extends Polygon {
2    constructor(length) {
3      // Here, it calls the parent class' constructor with lengths
4      // provided for the Polygon's width and height
5      super(length, length);
6      // Note: In derived classes, super() must be called before you
7      // can use 'this'. Leaving this out will cause a reference error.
8      this.name = 'Square';
9    }
10
11   get area() {
12     return this.height * this.width;
13   }
14
15   set area(value) {
16     this.area = value;
17   }
18 }
```

ACCESSOR **PROPERTIES**

Accessor properties were introduced in ES5 as a simplified way of providing getters and setters for JavaScript prototype functions

```
class Note {
    constructor(id, content, owner) {
        if (new.target === Note) {
            throw new Error('Note cannot be directly
        }

        this._id = id;
        this._content = content;
        this._owner = owner;
    }

    // read-only
    get id() { return this._id; }

    get content() { return this._content; }
    set content(value) { this._content = value; }

    get owner() { return this._owner; }
    set owner(value) { this._owner = value; }
}
```

CUSTOM
ERROR CLASSES

```
class InheritanceError extends Error { }


class Note {
    constructor() {
        if (new.target === Note) {
            throw new InheritanceError('Note cannot be directly constructed.')
        }
    }
}
```

CUSTOM ERROR
EXAMPLE

```
try {
    new Note(72, 'Vanilla note', 'benmvp');
}
catch (e) {
    // output: true
    console.log(e instanceof InheritanceError);
}
```

NO USE BEFORE DECLARATION

You cannot reference a class before it's declared