Rediscover
the meaning of technology

JavaScript Basics

ABOUT US

Plain Concepts is a Microsoft Certified Gold Partner and Plain Concepts won the Microsoft Partner of the Year 2016 in Spain and were a global finalist in the area of Data Platform

Microsoft | AI Partner

Microsoft Partner | Mixed Reality

MICROSOFT PARTNER OF THE YEAR 2016

12 MICROSOFT MOST VALUE PROFESSIONAL	XBOX PREMIER DEVELOPER PARTNER LIVE APPS	PIXELSENSE PREMIER PARTNE	MICROSOFT GOLD CLOUD PLATFORM	MICROSOFT SILVER COLABORATION AND CONTENT	MICROSOFT GOLD APP DEVELOPMENT
AGILE ALLIANCE CORPORATE MEMBER	MICROSOFT GOLD LIFECYCLE MANAGEMENT APPLICATION	MICROSOFT SILVER APP INTEGRATION	WINDOWS 8 APPLICATIONS PARTNER OF THE YEAR	XAMARIN PREMIUM CONSULTING PARTNER	ALM PARTNER OF THE YEAR FOR 7 CONSECUTIVE YEARS

OUR CLIENTS

Our customers are vast and cover a variety of business such as the tourism & hospitality, construction, energy, telecommunications, banking, media, communications and entertainment, creative agencies.

Microsoft · Telefónica · acciona · REPSOL · edp · Wonderware · BBVA · Sisteplant

CBRE · AENOR · groupm · Realmadrid · EA · ATRESMEDIA · DOLBY DIGITAL · Rakuten

du · AEGON · Santander · BBVA · PRISA BRAND SOLUTIONS · Fly Emirates · hp · ie Reinventing Higher Education

ALSTOM · CBRE · SGS · TÜV

OUR EXPERIENCE

2006

We are an award-winning thech company founded in 2006 by 4 Microsoft MVPs.

250+

Plain Concepts has more than 250 employees and we are still growing.

10+

We are experts in the most advanced technologies.

1000+

Over 1,000 projects behind us an industry benchmark.

8

We are based in 8 global offices, in 3 different continents.

JavaScript Basics

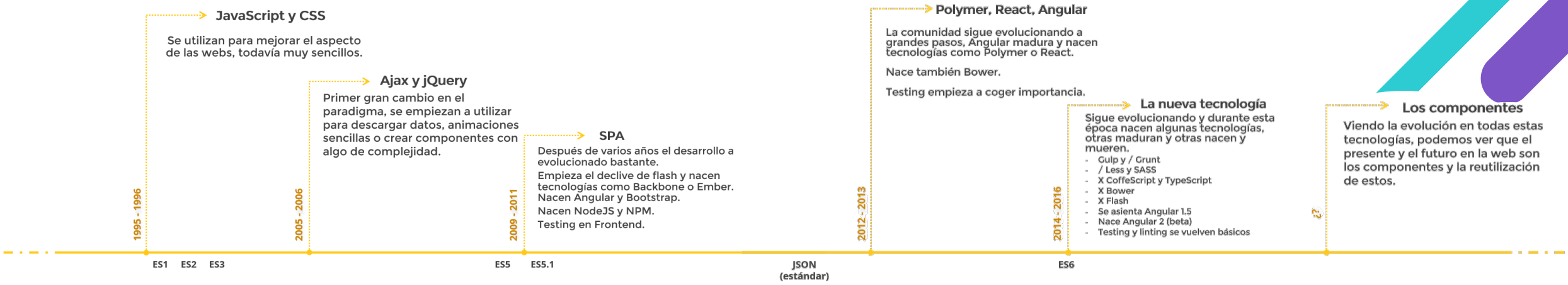
Fundamentals

Basics

Advanced

plain concepts

Introduction to JavaScript



Variables

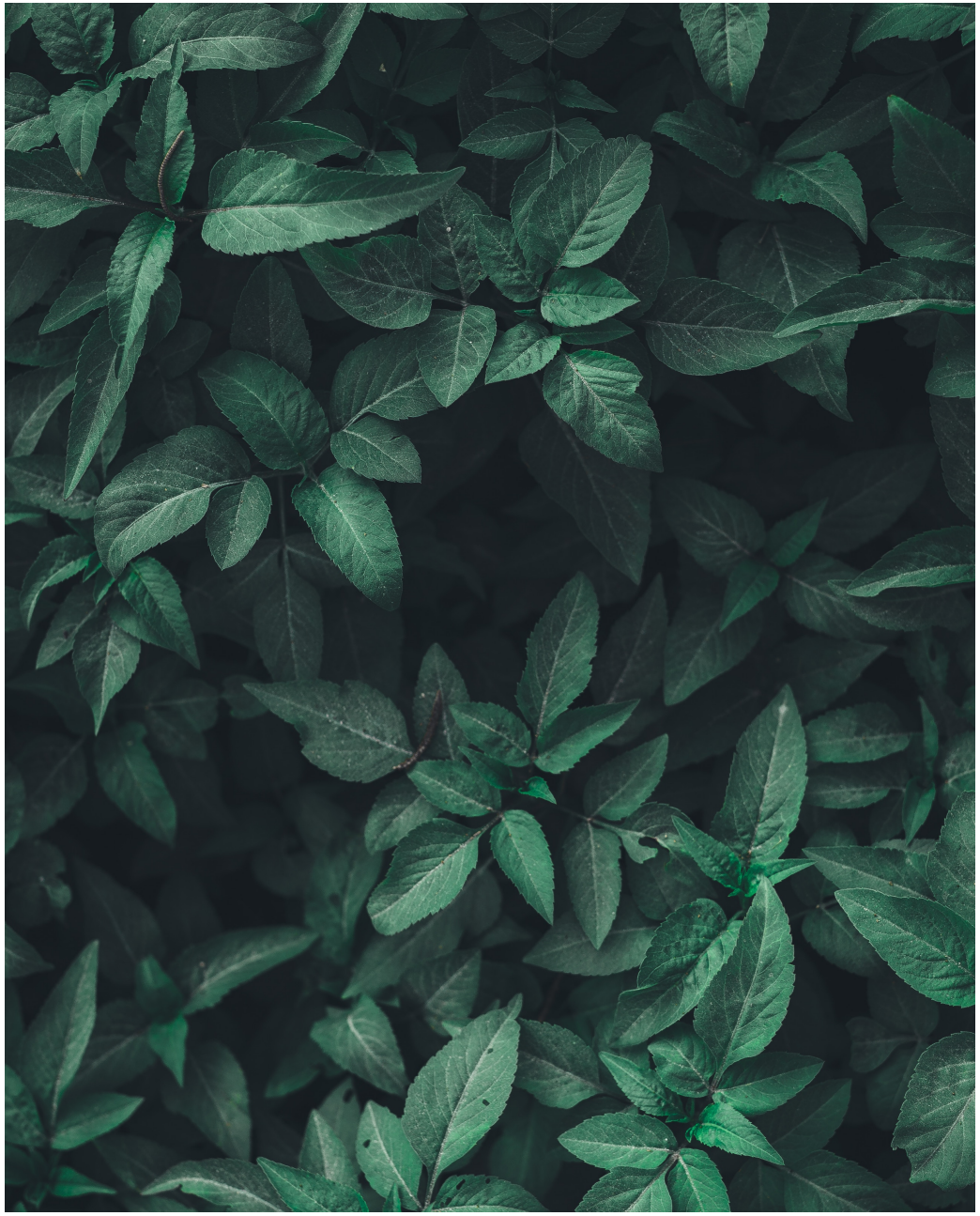
Difference between:

- Var

- Let

- Const

```
var a = "Hello";
const b = "world";

if (true) {
  let c = true;
}

console.log(a, b, c);
```



plain concepts

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)

console.log(`hello ${"world"}`);
```

Data types

- Number
- String
- Boolean
- Null & Undefined
- Object & Symbol & Function
- String Template

plain concepts

```
String(true)
true + ''
`${true}`

Number('1')
'1' / '2'
Number.parseInt('15', 10)
Number.parseInt('F', 16)
Number.parseFloat('15.1')
15.1.toFixed(3) // '15.100'

Boolean('')
Boolean(' ')
Boolean('true')
Boolean('false')
```

Conversion of types

- ToString
- ToNumber
- ToBoolean

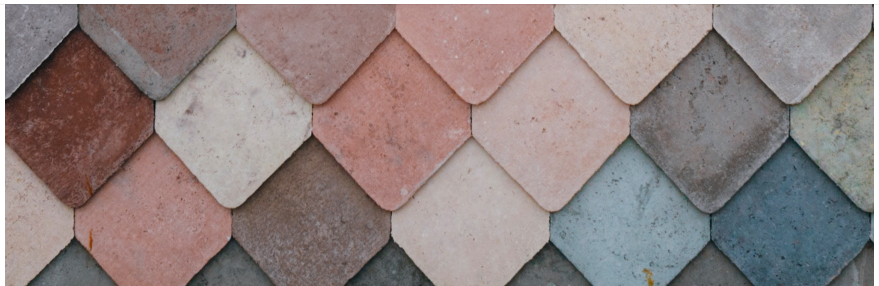
plain concepts

Operators

- Unary
- Binary
- Concatenation (Binary +)
- Assigment =
- Remainder %
- Exponentiation **
- Increment ++ and decrement –
- Bitwise operators (& | ^ ~)

```
let a = 1
a = -a

let b = 1
let c = 2
const d = b * c

'a' + '2'

a = b = c = 2 + 2

const x = 4 % 2
const y = 2 ** 3

a = 1
a++
++a
```

```
3 == 4
2 >= 2

'b' > 'a'
'ab' > 'aaaaa'

1 == '1'
1 === 1

null == false
undefined === null

null > 0
null == 0
null >= 0  // :0
```



Comparisons

- Numbers
- String
- Strict equality
- Null & undefined
- Strange behaviours

plain concepts

Conditional operators

- If
- Boolean
- Ternary

```
if (1 === 1) {
}

if (true) {
}

const a = 1 === 1 ? 'ok' : 'bad'
```

plain concepts

Logical Operators

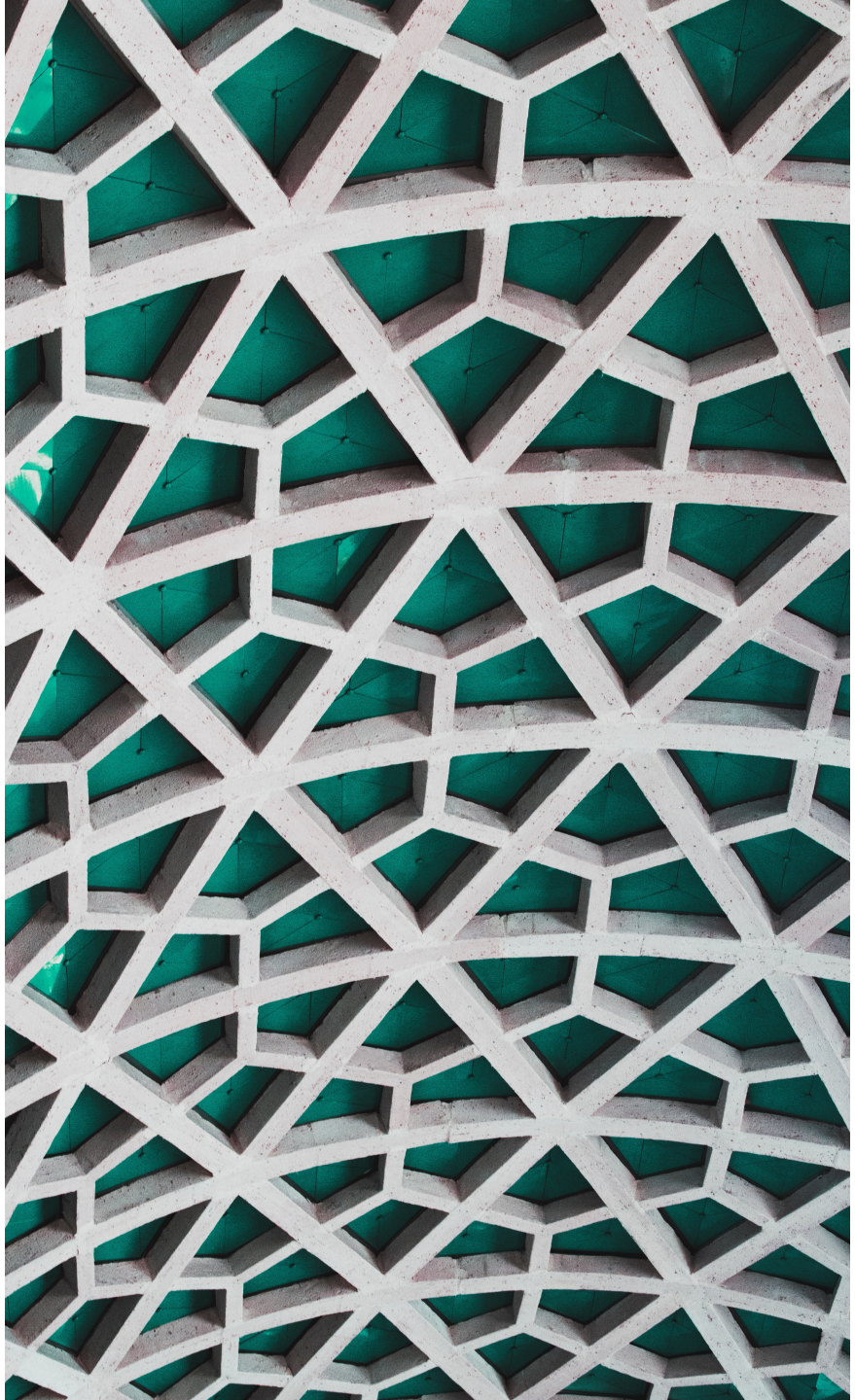
- OR | |
- AND &&
- NOT !
- DOUBLE NOT !!

```
const a = 1 || 2 // Truthy

const b = true && false

const c = !false

const d = !!1
```

Loops: while and for

- While
- Do..While
- For
- Breaking a loop
- Continue (light break)

```
while (a > 2) { ... }

do { ... } while (a > 2)

for (let i = 0; i < 20; i++) {

  if (i === 2) {
    continue;
  }

  if (i > 4) {
    break;
  }
}
```



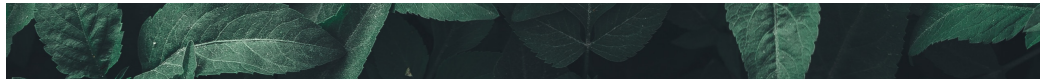
The Switch statement

(We can usually change it for a pattern or a dictionary)

```
let a = 2 + 2;

switch (a) {
  case 3:
    console.log('Hello');
    break;
  case 4:
    console.log('World');
    break;
  default:
    alert( "I don't know such values" );
}
```

Functions

- Function declaration
- Context
- Parameters
- Return values
- Avoiding else in functions



```
function hello() {
  console.log('world');
}

const myFunction = hello() {
    console.log(world);
}

let a = 1;
function b() {
  let a = 2;
}

function c(a, b = 1) { ... }

function d() {
  return 1;
}

function e(a) {
  if (!a) {
    return;
  }

  return a;
}
```

plain concepts

Objects and this

EVERYTHING IS AN OBJECT!

```
const myObject = {
    a: 1,
    b: 'hello',
    c: function world() { ... }
}

delete myObject.b
```

```
this.a = 1;
const b = {
    a: 2,
    bb: function() {
        console.log(this.a)
    }
}

function c() {
    return  function() {
        return function() {
            console.log(this.a)
        }
    }
}
```

plain concepts

Other Methods

- String methods
- Number methods

```
'hello'.toUpperCase()

9.23.toFixed(4)
```

plain concepts

```
let a = new Array(2)
let b = [null, null]

for (let char of 'test') {
  console.log(char);
}

const user = {
  a: 'aa',
  b: 'bb'
}

Object.values(user)
Object.keys(user)
Object.entries(user)

const date = new Date()
Date.now()
```

More Data types

- Arrays
- Iterables
- Object.keys, .entries, .values
- Date

plain concepts

Map, Set, WeakMap, WeakSet

Map is a collection of keyed data items, just like an Object. But the main difference is that Map allows keys of any type (Dictionaries).

A **Set** is a collection of values, where each value may occur only once (Lists).

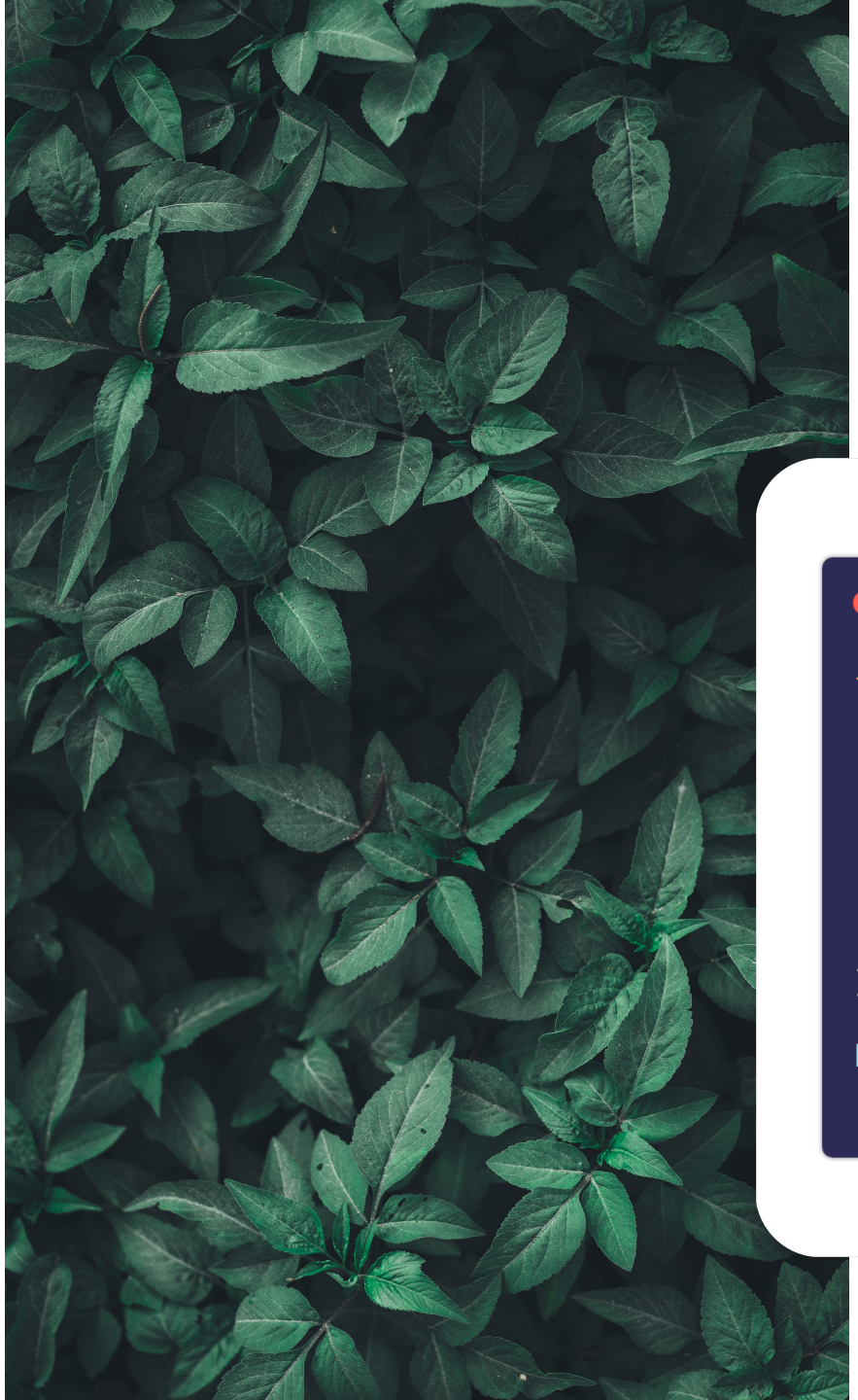
The **Weak'** are used to have same object reference in the key.

```
let recipeMap = new Map([
  ['cucumber', 500]
]);

let set = new Set();
let john = { name: "John" };
let pete = { name: "Pete" };

set.add(john);
set.add(pete);

console.log(set.size);

for (let user of set) {
  alert(user.name);
}

let weak = new WeakSet();
weak.add(john);
weak.has(john)
john = null;
weak.has(john)
```

JSON Methods & Try / catch

- Stringify
- Parse
- Try Catch and throw

```
JSON.stringify({})

JSON.parse('{}')

// Be careful
try {
  JSON.parse('{]]]]')
} catch(e) {
  console.log('error')
} finally {
  console.log('finished')
}

function a() {
  throw new Error('Error')
}
```



Recursion

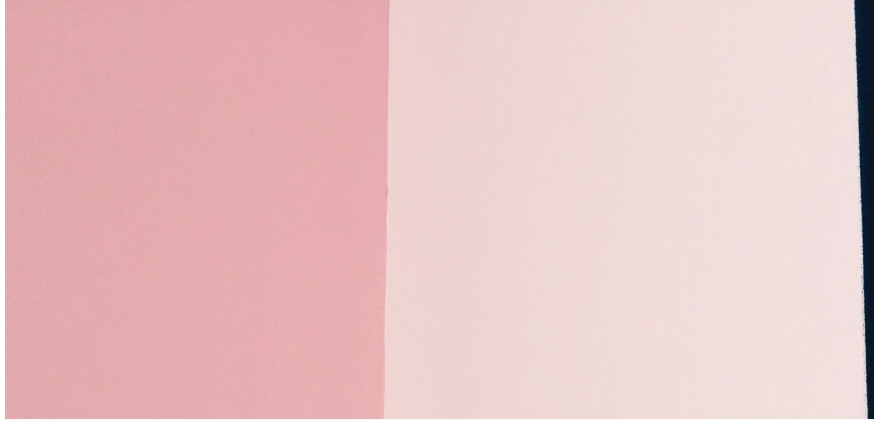
```
function hello(n = 0) {
  console.log(n);

  if (n === 10) {
    return;
  }

  hello(++n);
}

hello()
```

plain concepts

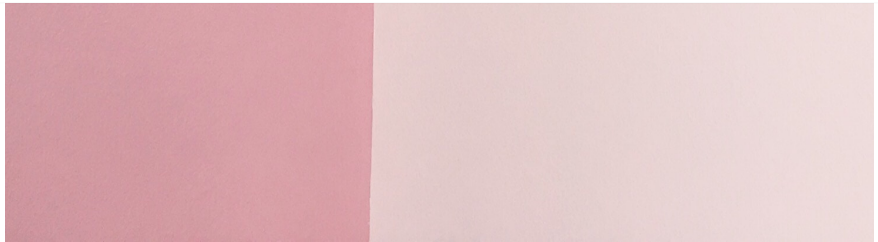
Rest params and spread

- Rest params instead of arguments
- Spread operator in Arryas
- Spread operator in Objects

```
function hello(a, ...b) {
    console.log(a, b);
}

hello(1, 2, 3, 4, 5)

const b = [3, 4]
[1, 2, ...b]

const c = { a: 1 }
{ b: 2, ...c }
```



```
function makeCounter() {
  let count = 0;

  return function() {
    return count++;
  };
}

const counter = makeCounter()
counter()
counter()
```



Closure

Creating private contexts to use in our functions.

Bind, call and apply

Pasing context to functions

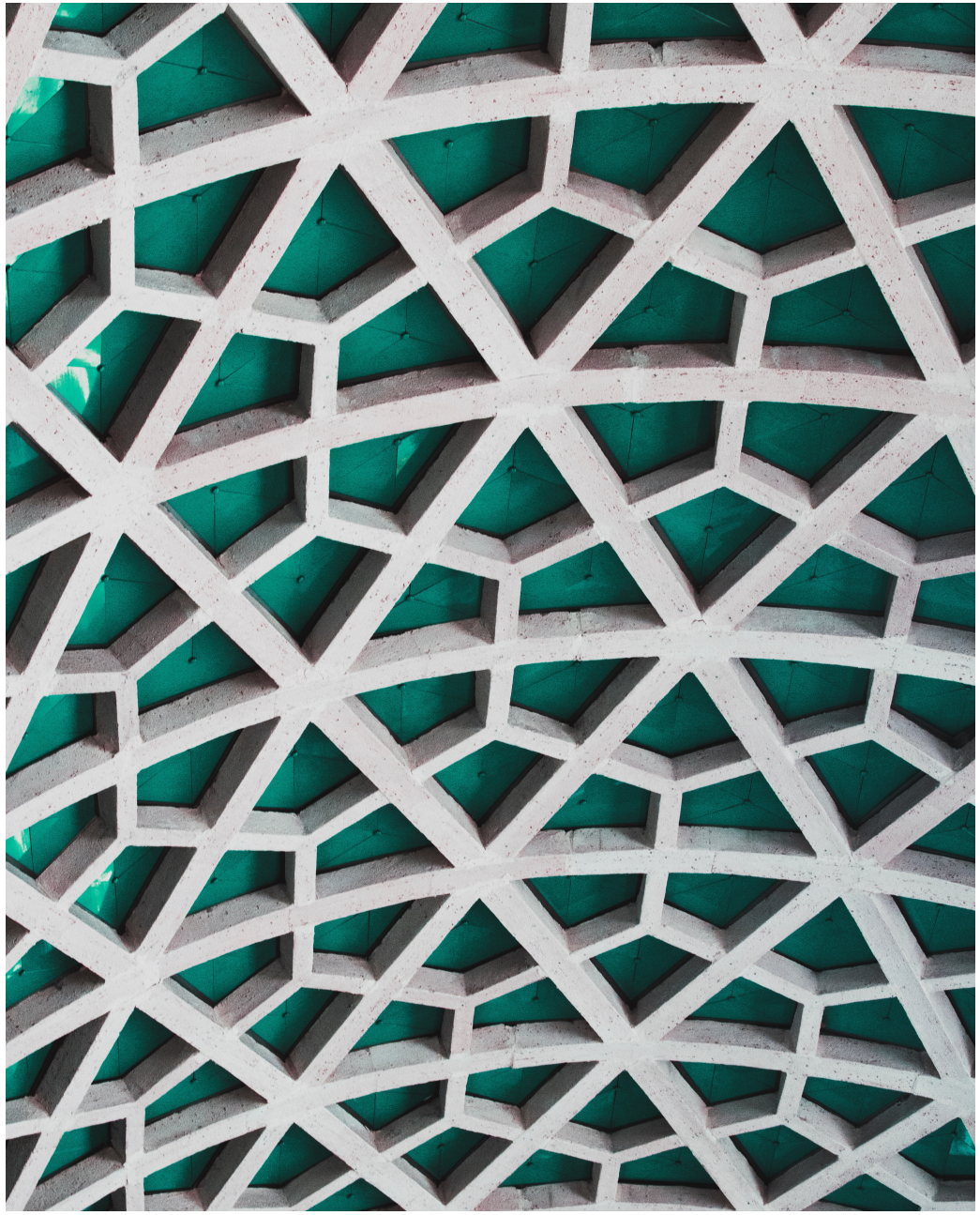
```
this.a = 1;
const b = {
    a: 2,
    bb: function(n = 1, x = 1) {
        console.log(this.a * n * x)
    }
}

b.bb.bind(this)()
b.bb.call(this, 10, 20)
b.bb.apply(this, [10, 20])
```

Arrow functions

The evolution of functions as parameter.

Anonymous functions with parent context.

```
this.a = 1;
const b = {
    a: 2,
    bb: () => {
        console.log(this.a)
    }
}

b.bb()
```

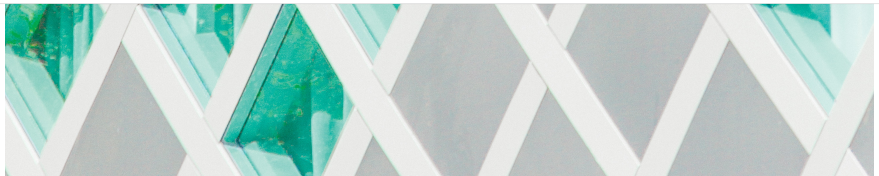


plain concepts

```
function n(x) {
  console.log(x)
}

setTimeout(() => n(1), 10)

n(2)
const t = setInterval(() => n(2), 1000)

function n(x) {
  console.log(x)
  requestAnimationFrame(() => n(x))
}

requestAnimationFrame(n(1))

clearInterval(t)
```

Timers

- SetTimeout
- SetInterval
- RequestanimationFrame (bonus*)



plain concepts

```
class B {
    constructor() {
      console.log(1)
    }
}

class A extends B {
    constructor() {
      super()

      this.a = 1
    }

  static alert(x) {
    console.log(x)
  }

  get prop() {
    return this.a + 10
  }

  set prop(x) {
    this.a = x - 5
  }
}
```

Classes

Introduction

Inheritance

Static

Getters and Setters

Promises

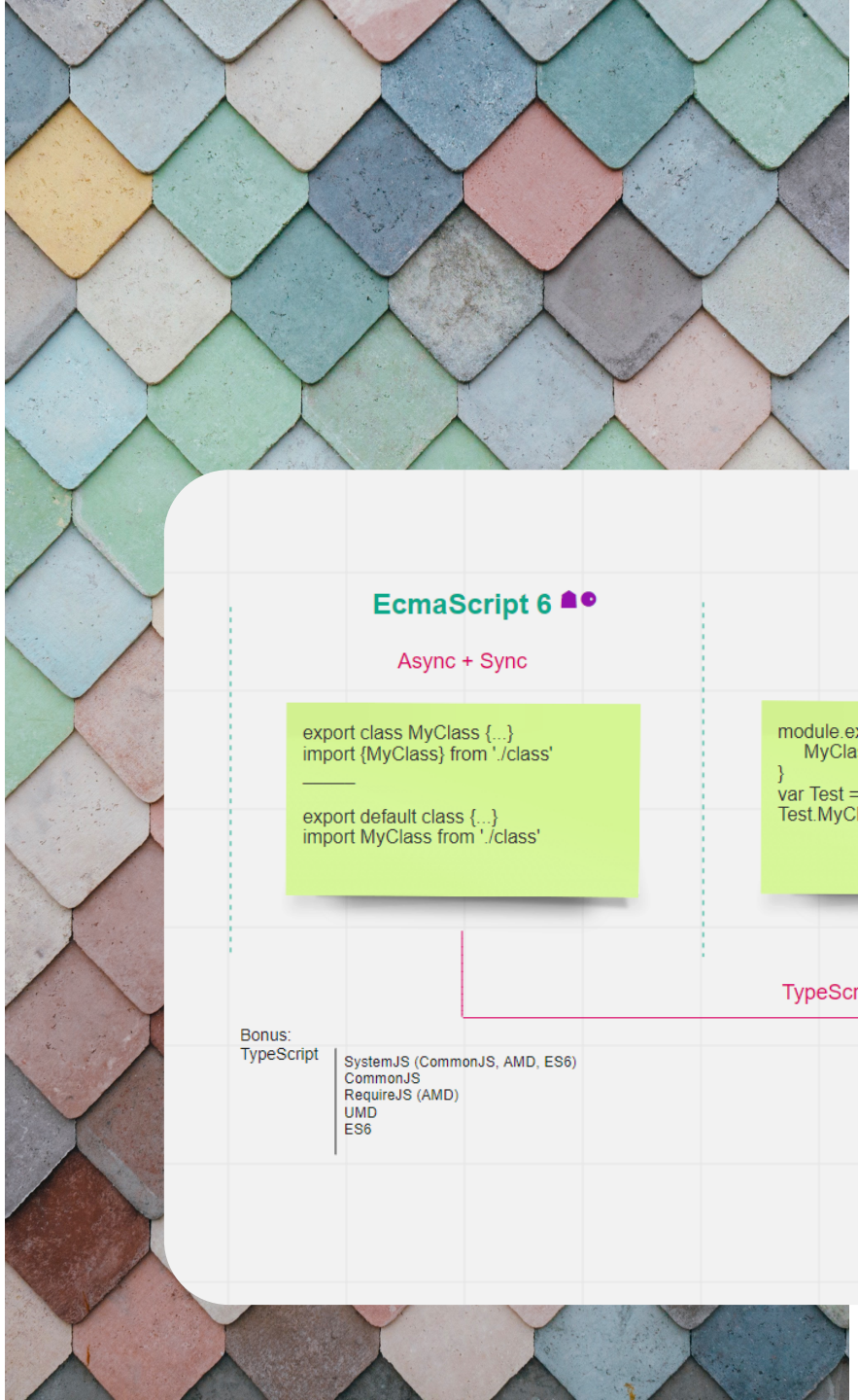
- new Promise
- Then
- Catch
- Finally

```
function test(n = 0) {
  return new Promise((resolve, reject) => {
    if (n > 10) {
      reject()
      return
    }

    resolve()
  })
}

test(5)
  .then(() => console.log('ok'))
  .catch(() => console.log('error'))
  .finally(() => console.log('end'))
```

```
(async () => {
  function test(n = 0) {
    return new Promise((resolve, reject) => {
      if (n > 10) {
        reject()
        return
      }

      resolve()
    });
  }

  try {
    const t = await test(5)
    console.log('ok')
  } catch (e) {
    console.log('error')
  } finally {
    console.log('end')
  }
})()
```

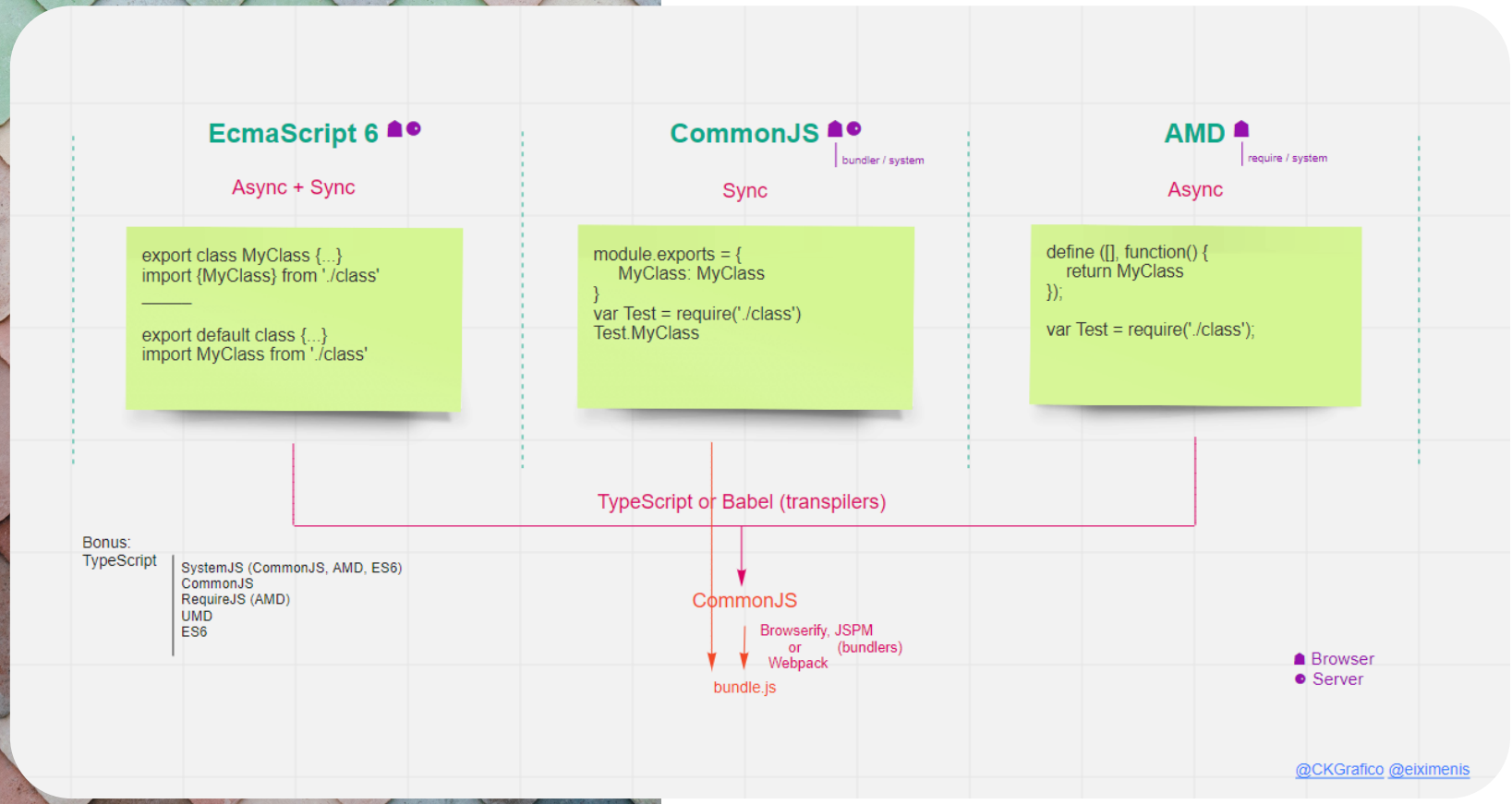
Async await

Next Promises step



Modules

bit.ly/js-modules-board



plain concepts

Generators

Returning multiple values, one for after another on-demand.

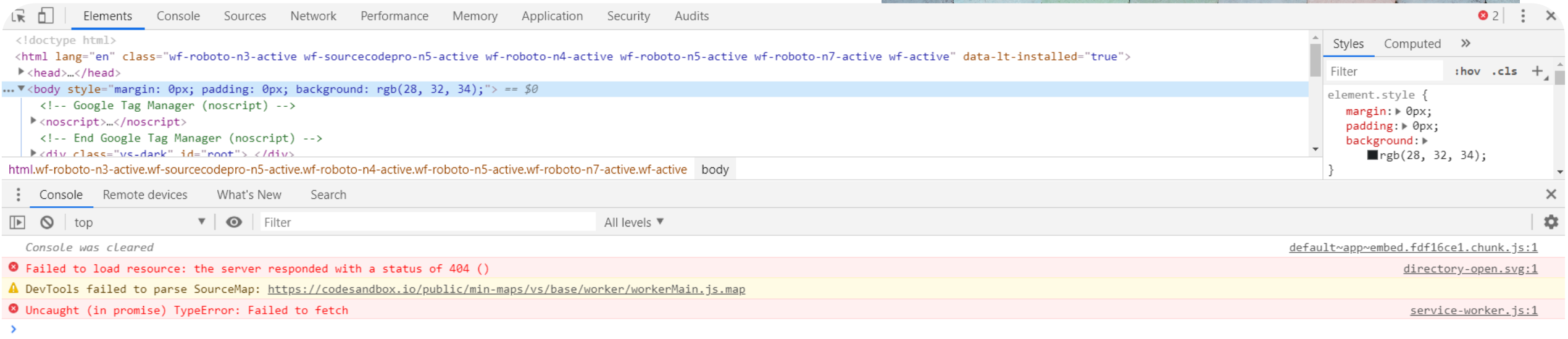
```
function* generateSequence() {
  yield 1
  yield 2
  return 3
}

const generator = generateSequence()
console.log(generator.next())
console.log(generator.next())
console.log(generator.next())
console.log(generator.next())
```

Currying

f(a, b, c) -> f(a)(b)(c)

```
// const curry = f => a => b => c => f(a, b, c)
function curry(func) {
  return function (a) {
    return function (b) {
      return function (c) {
        return func (a,b,c);
      }
    }
  }
}

function log(date, importance, message) {
  console.log(`[${date.getHours()}:${date.getMinutes()}] [${importance}] ${message}`)
}

const _log = curry(log)
log(new Date(), 'DEBUG', 'some debug')
_log(new Date())('DEBUG')('some debug')

const todayLog = _log(new Date())
todayLog('DEBUG')('some debug')
todayLog('ERROR')('some error')
```

Browser Debugging tools



plain concepts

JavaScript and the DOM

- Get elements and Nodes
- Events
- Modify the dome
- Libraries and UI frameworks
- Virtual DOM concept
- Components

Linting

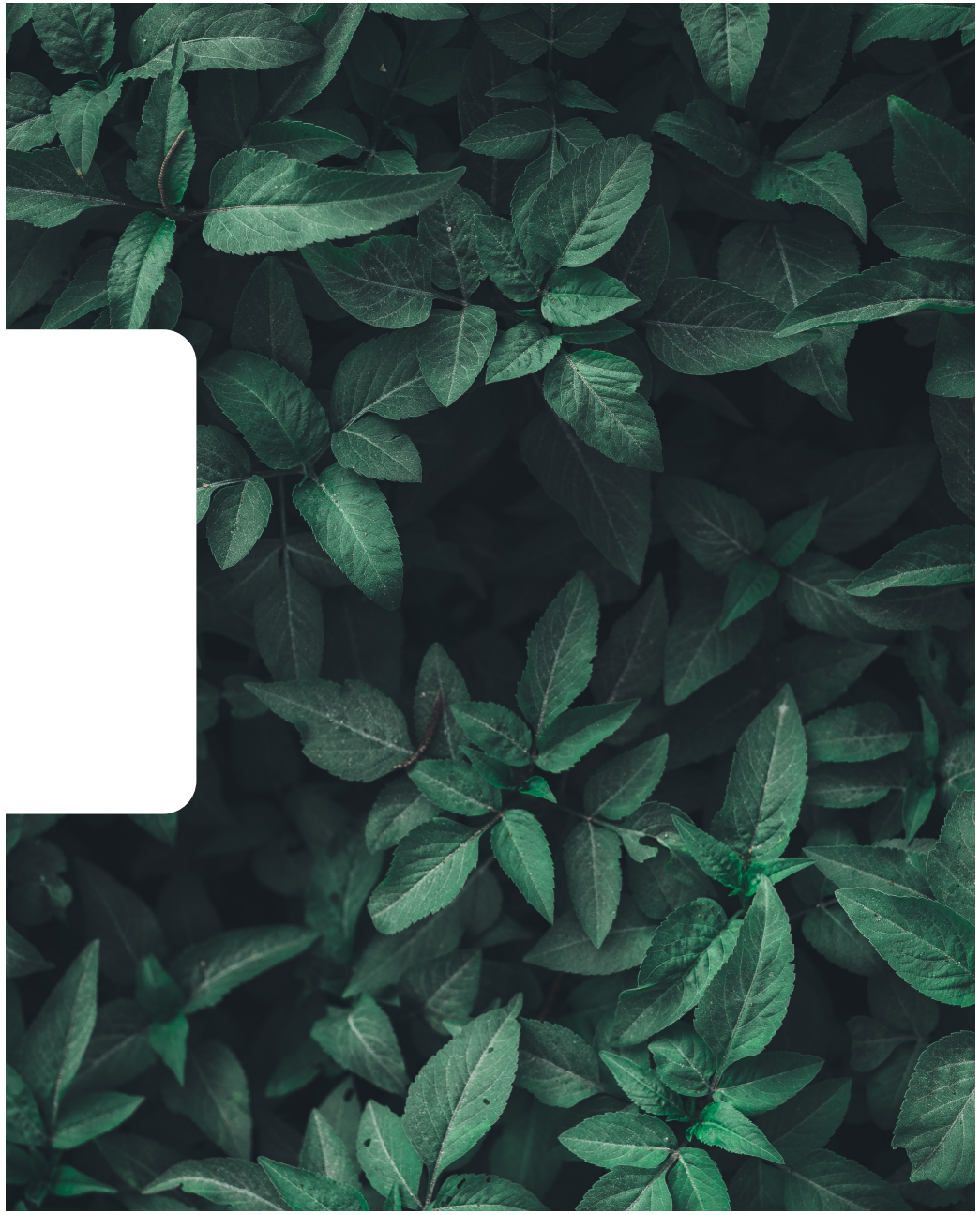
ESLint

Custom rules

Automation of Tasks (npm, webpack, etc.)

webpack

Unit Testing



UI Testing

cypress

plain concepts

SINGLETON

Patrón de instancia única

Transient

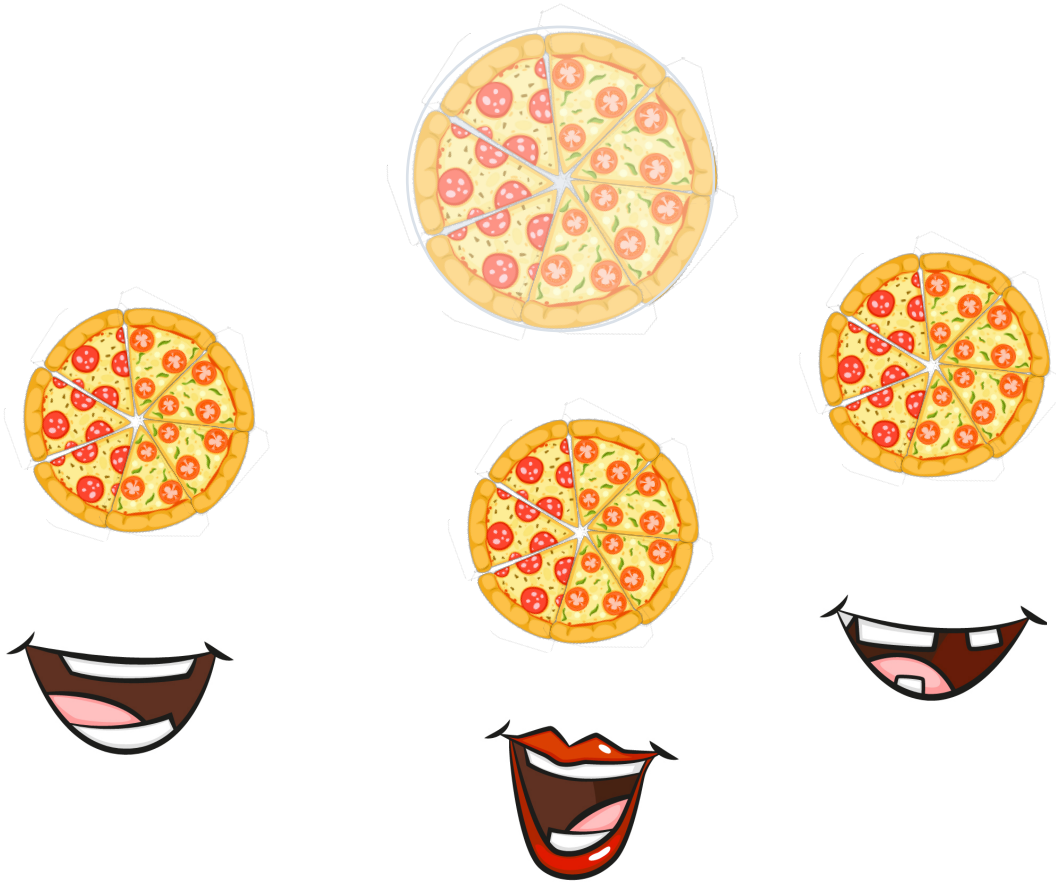
Singleton

Freepik vectors

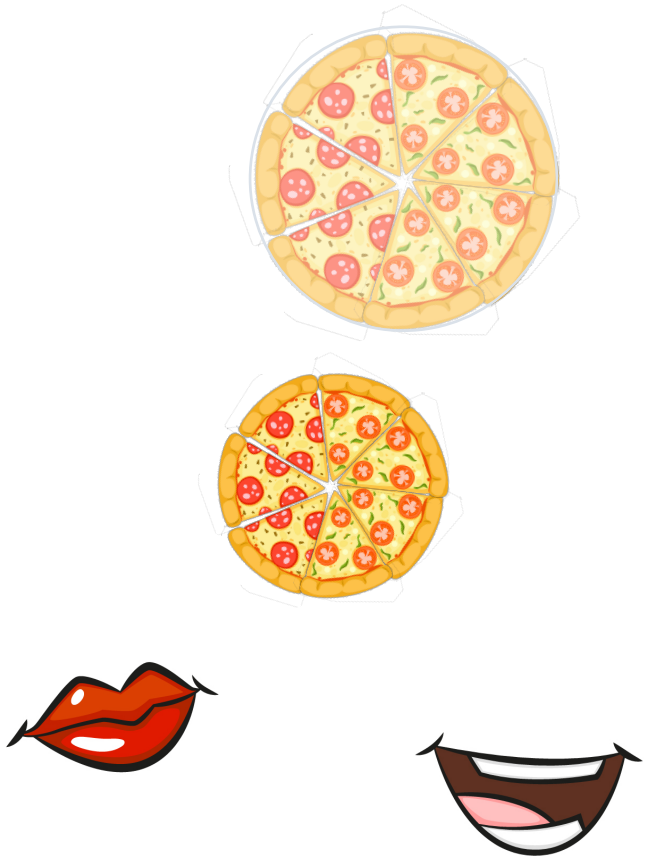
SINGLETON

Patrón de instancia única

Transient



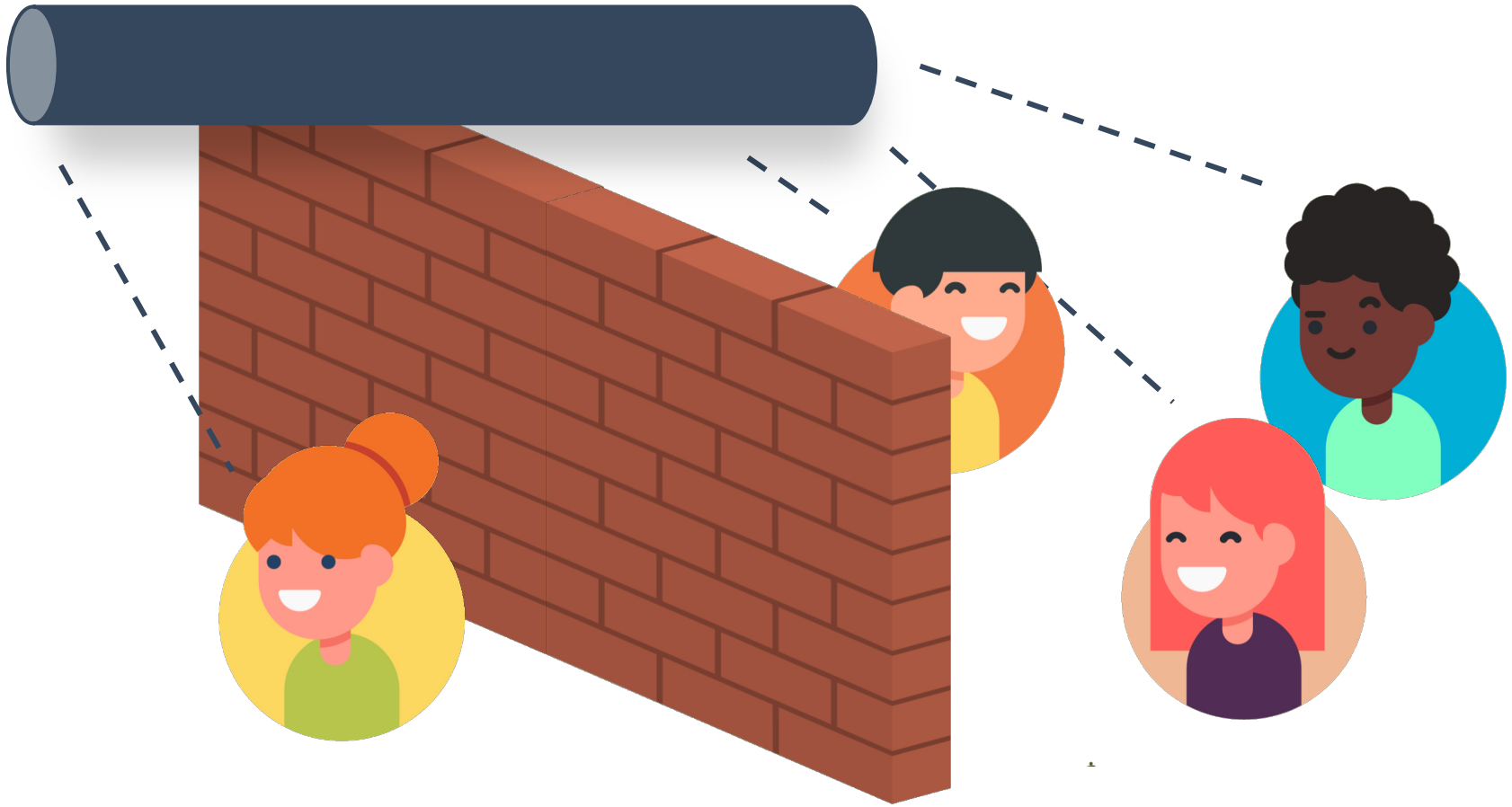
Singleton



Freepik vectors

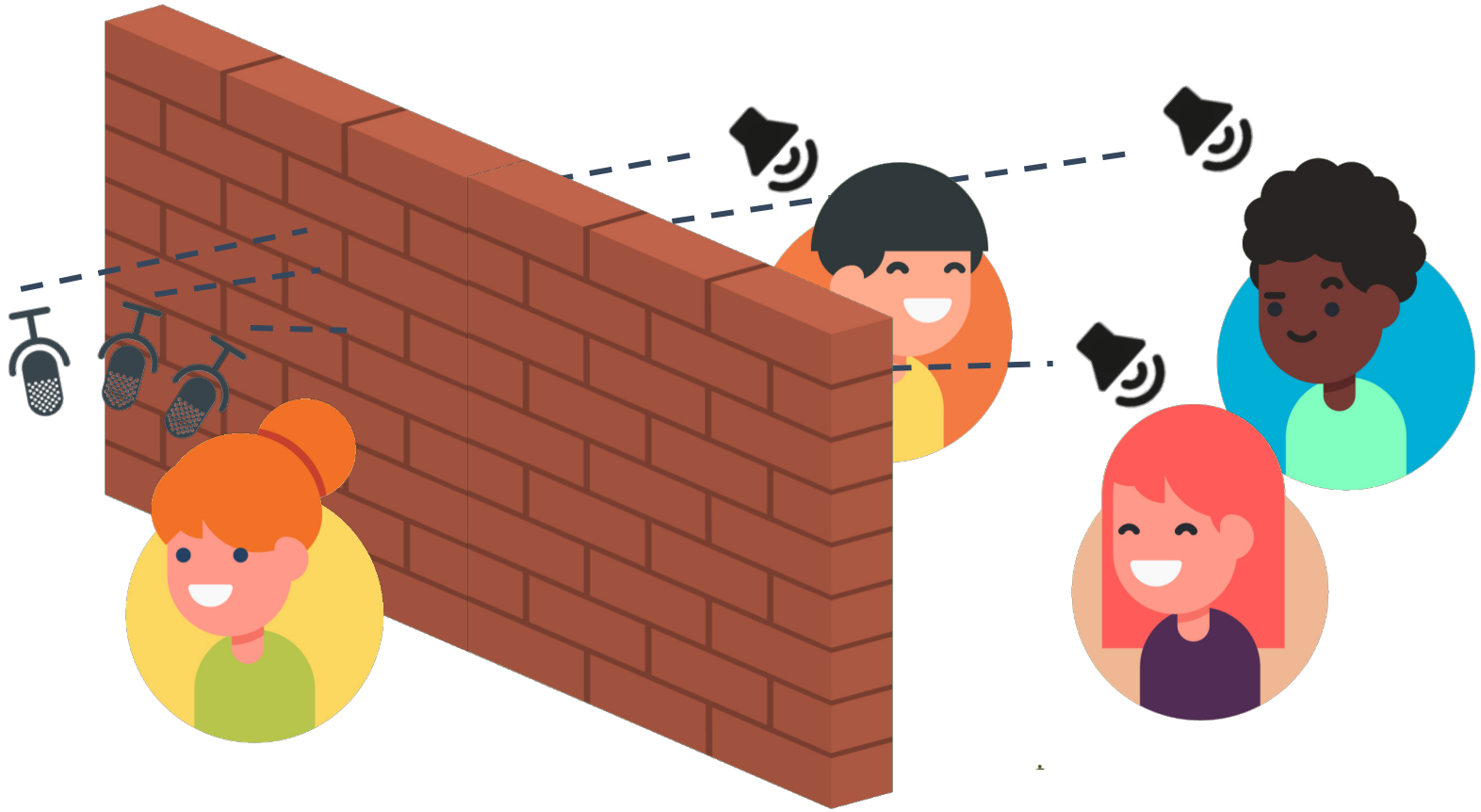
PUBLISH / SUBSCRIBE

Patrón publicación / subscripción



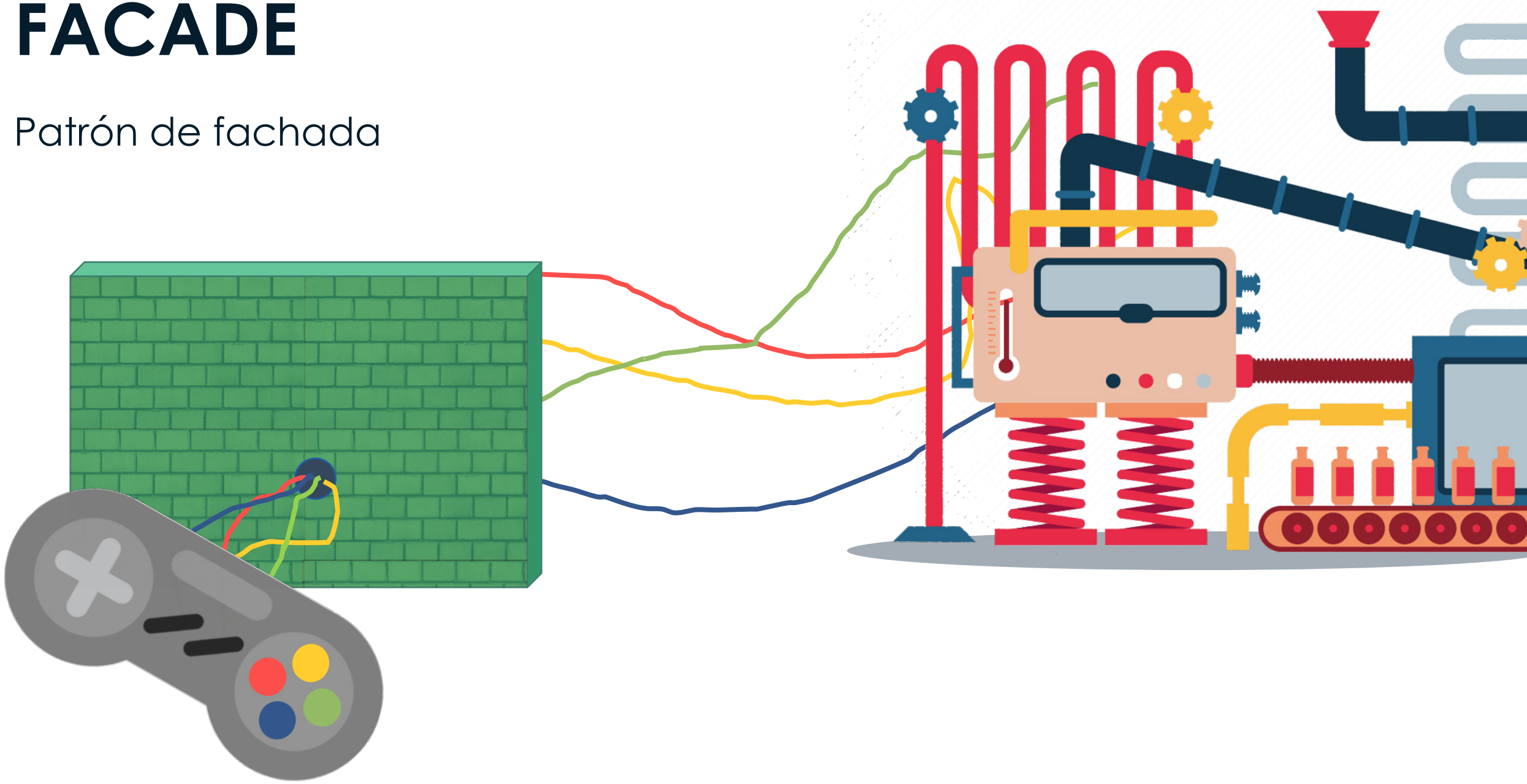
OBSERVER

Patrón de observables



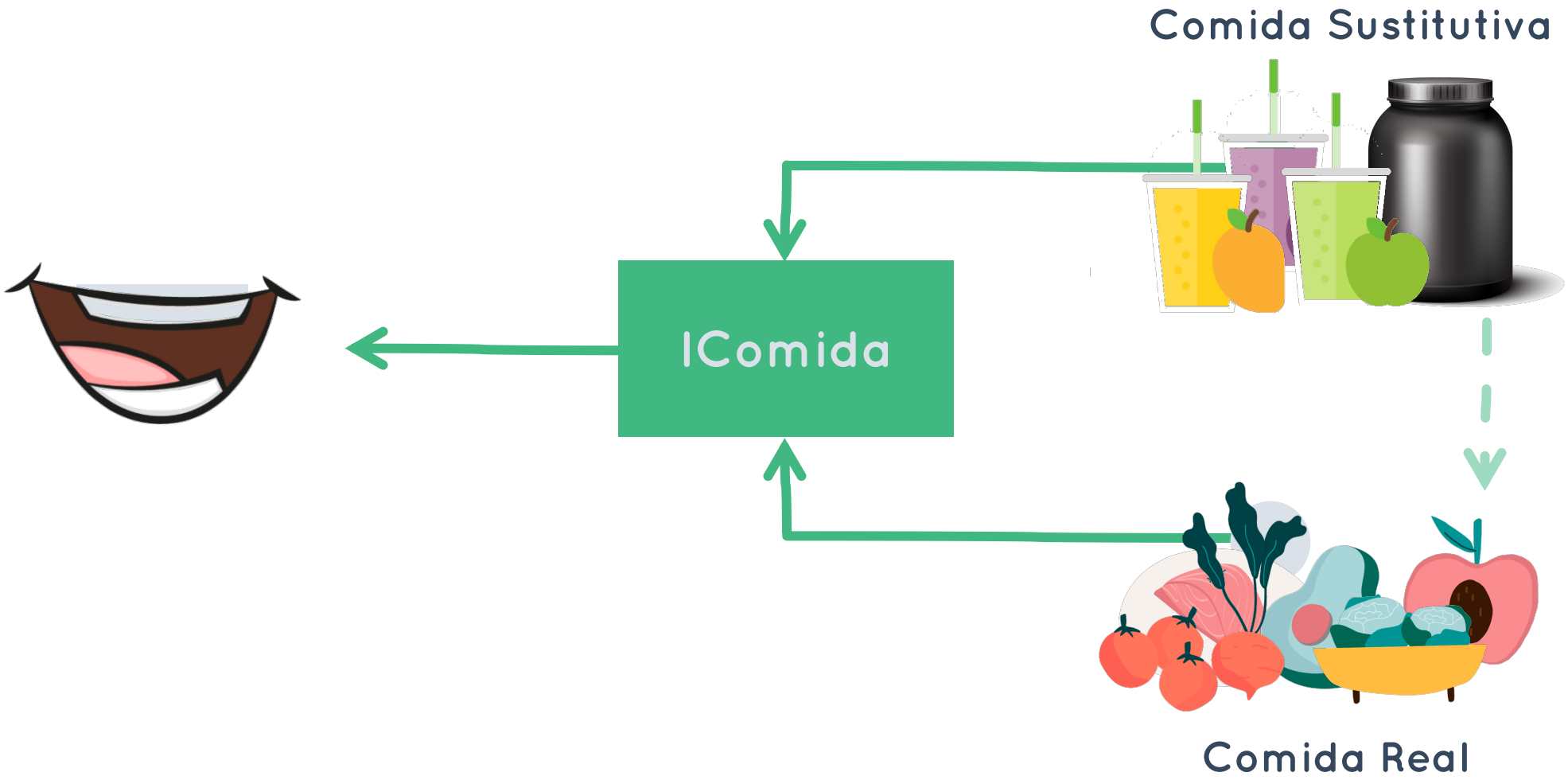
FACADE

Patrón de fachada



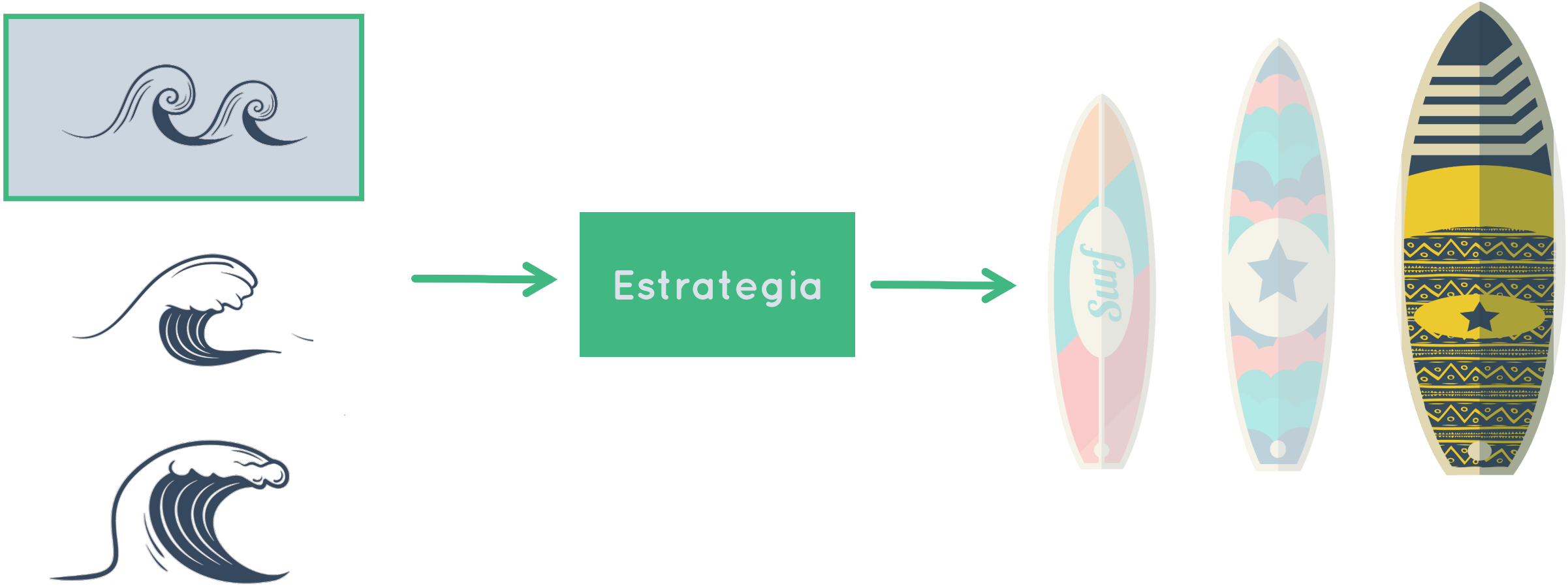
PROXY

Patrón de ‘intermediario’



STRATEGY PATTERN

Patrón de estrategia

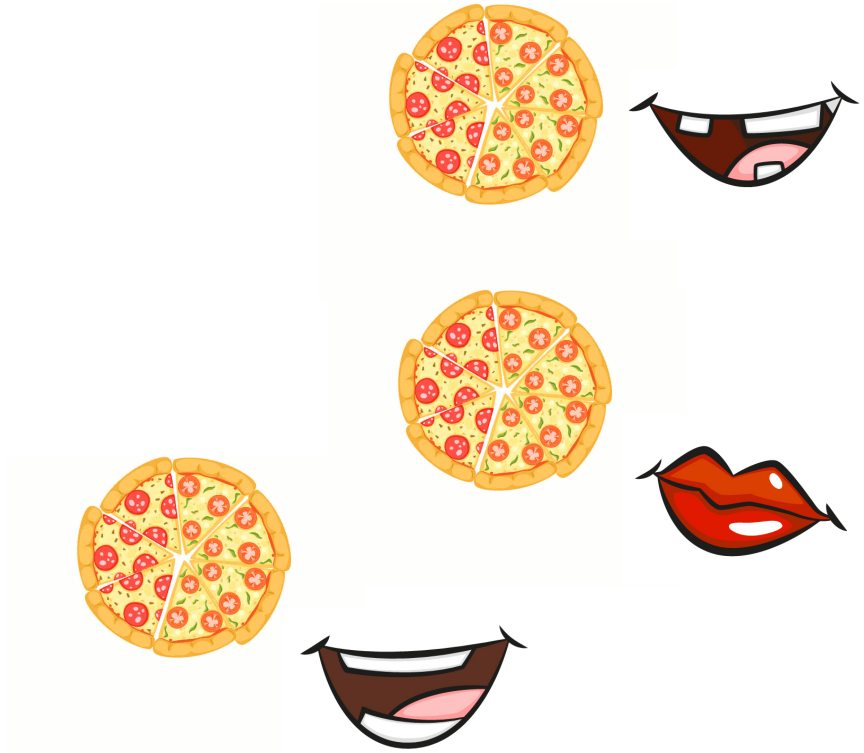


DEPENDENCY INJECTION

Patrón de inyección de dependencias

Singleton

Services



plain concepts

Rediscover
the meaning of technology

MADRID

BARCELONA

BILBAO

SEVILLA

LEÓN

CORUÑA

LONDON

FRANKFURT

www.plainconcepts.com

For further information

info@plainconcepts.com

AMSTERDAM

SEATTLE

DUBAI