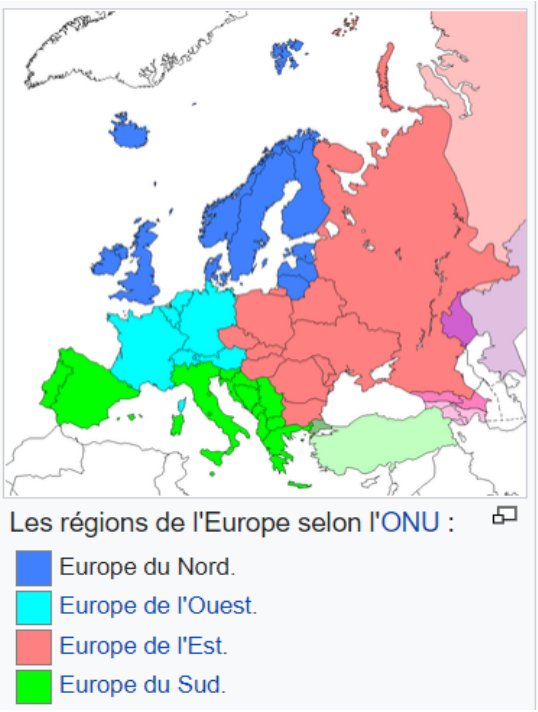


Nous récupérons le résultat d'une base de données contenant un liste de pays à afficher : https://output.jsbin.com/lanasay.js

Ce fichier est un tableau contenant les pays dans l'ordre alphabétique.

[

" 📍 Allemagne ",

" 📍 Belgique ",

" 📍 Suède "

]

🚀Objectifs

L'objectif premier est de trouver la zone géographique d'un pays européen.

Par exemple, pour l'Allemagne la zone géographique est
📍 Europe de l'Ouest .

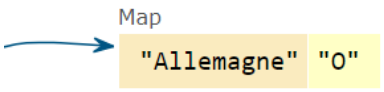
MAP

Nous allons découvrir la puissance de Map, au travers d'un certain nombre d'exemples.

Définition

L'idée de Map est d'associer au travers d'un dictionnaire une valeur à une autre.

Par exemple, on peut associer "Allemagne" a sa zone géographique "O" (Ouest).



"Allemagne" est appelé une clé (key)

"0" est appelé une valeur (value)

Commençons par définir un dictionnaire[1] **countryZone** basique pour les 27 pays.

La création d'un dictionnaire est new Map().set(key,value)...

```
const countryZone = new Map()
        .set("Allemagne","O")
        .set("Belgique","O")
        ...
        .set("Suède","N");
```


[1] dictionnaire = Map en js.

Une autre définition est possible avec new Map([[key,value],...])

```
const countryZone = new Map([
        ["Allemagne","O"],
        ["Belgique","O"],
        …
        ["Suède","N"]
])
```

Une autre définition est possible avec Object.entries qui renvoie un tableau de tableau.

```
const countryZone = Object.entries({
   "Allemagne": "O",
   "Suède": "N"
})

console.log(countryZone)
> Array [Array ["Allemagne", "O"], Array ["Suède", "N"]]
```

finalement nous pouvons écrire.

```
const countryZone = new Map(Object.entries({
   "Allemagne": "O",
   "Belgique": "O",
…
   "Suède": "N"
}))
```

Renvoyer un élément

La méthode get permet de trouver la valeur d'une clé.

Ainsi pour trouver la zone d'un pays on écrit :

```
const ZoneAllemagne = countryZone.get("Allemagne");
```

code

On pourrait tester dans **node** le code

```
const obj = {
    "France": "O",
    "Suède": "N"
}

const  abbreviation = new Map(Object.entries({
   "O": "de l'Ouest",
   "E": "de l'Est",
   "S": "du Sud",
   "N": "du Nord",
}));

for (let [pays, zone] of Object.entries(obj)){
  console.log(`la ${pays} est en europe ${abbreviation.get(zone)}`);
}
> "la France est en europe de l'Ouest"
> "la Suède est en europe du Nord"
```

Renvoyer les clefs et valeurs

Retrouvez tous les pays équivaut à retrouver les clés. Nous utilisons la méthode keys().

Nous devons également utiliser une méthode Array.from car Keys() renvoie un objet iterator et non un tableau.

La méthode Array.from() permet de créer une nouvelle instance d'Array (une copie superficielle) à partir d'un objet itérable ou semblable à un tableau.

Les pays sont donnés par :

```
Array.from(mapEurope.keys())
```

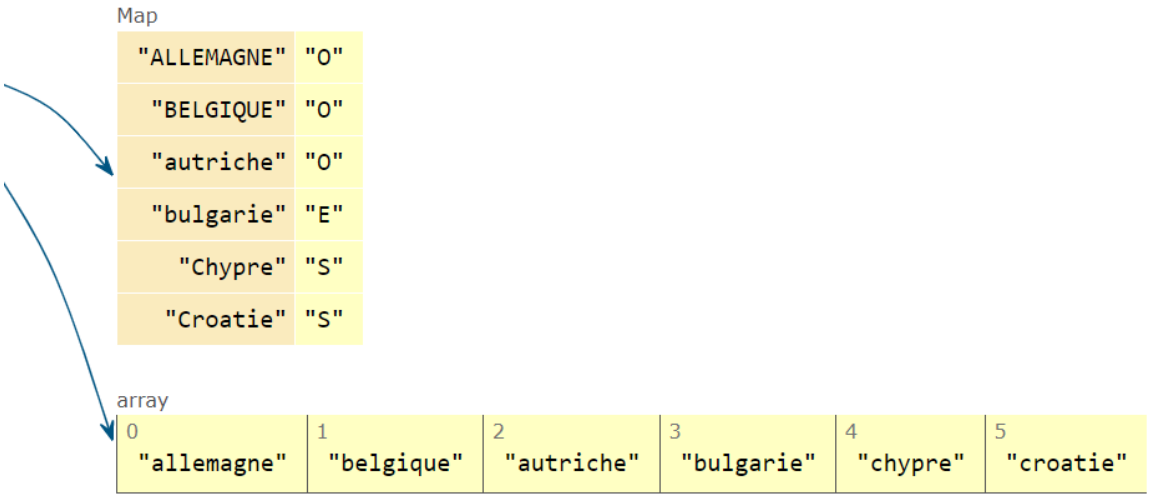
SOS code

Array.from permet d'utiliser également une fonction de **callback.**

Exemple :

Remarquez que les clés ont des écritures non normalisées. La fonction de callback (forEach) va normaliser l'écriture des clés.

```
Array.from(mapEurope.keys(), country => country.toLowerCase());
```



SOS code

La fonction de callback dispose de deux paramètres.

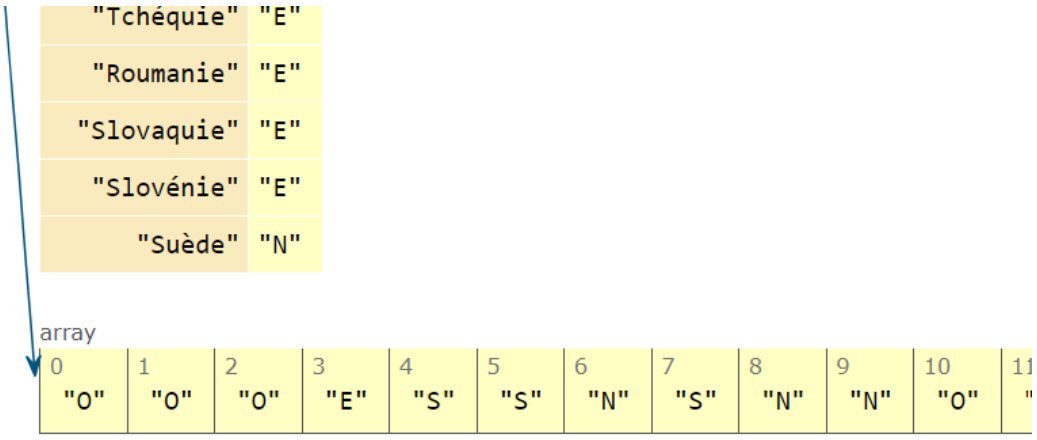
```
let countries = Array.from(
  mapEurope.keys(),
  (country, i) => `${i} : ${country.toLowerCase()}`
);
```

code

Si la méthode keys() permet de trouver les clés, la méthode values() permet de trouver les valeurs.

Ainsi les zones sont données par

```
Array.from(mapEurope.values())
```



code

Bilan

```
const zonesTab = Array.from(mapEurope.values());
const countriesTab = Array.from(mapEurope.keys());
const countryZoneTab  = Array.from(mapEurope.entries());
```

code

SET

Set est un ensemble à valeur unique.

Définition

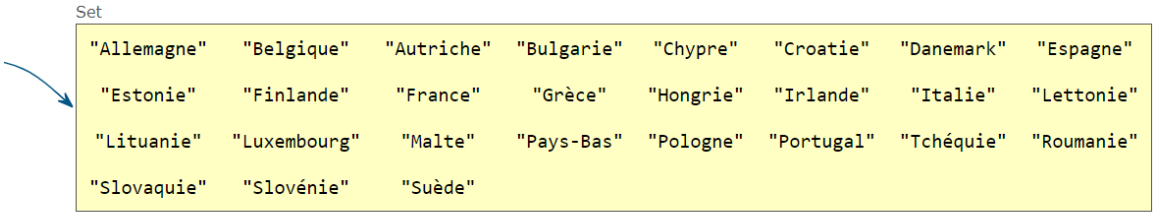
new Set([]) permet de définir un ensemble.

Voici l'ensemble des pays européens.

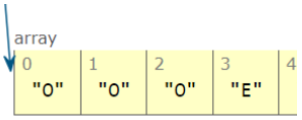
let paysEurope = new Set(

["Allemagne","Belgique","Autriche","Bulgarie","Chypre","Croatie","Danemark","Espagne","Estonie","Finlande","France","Grèce","Hongrie","Irlande","Italie","Lettonie","Lituanie","Luxembourg","Malte","Pays-Bas","Pologne","Portugal","Tchéquie","Roumanie","Slovaquie","Slovénie","Suède"]

);



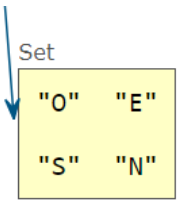
Les valeurs de l'ensemble seront uniques.

Ainsi, nous pouvons réduire les doublons du code *mapEurope.values()* pour obtenir un ensemble unique de zone. 

...

let zones = new Set(Array.from(mapEurope.values()))[2];

[2] une autre écriture : forEach Map



code

mapEurope.values() est un itérateur. Cela suffit à Set pour construire un ensemble.

Ainsi à partir de Map, on peut obtenir l'ensemble des zones très facilement avec simplement

```
const zones = new Set(mapEurope.values());
```

code

Bilan

Il est très facile d'obtenir les zones et pays à partir d'un dictionnaire.

```
let zones = new Set(mapEurope.values());
let countries = new Set(mapEurope.keys());
let ensemble = new Set(mapEurope.entries());
```

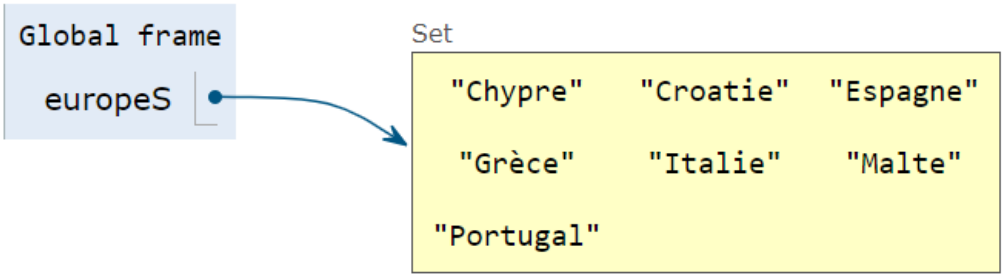
code

Objectifs

Nous allons rassembler dans quatre ensembles (N,S,E,O) les pays d'une même zone.

L'exemple suivant définit l'ensemble de la zone S (Sud).

```
let europeS = new Set(
["Chypre","Croatie","Espagne","Grèce","Italie","Malte","Portugal"]
);
```



Nous avons défini les pays de la zone Sud. Nous voulons regrouper **"automatiquement"** les pays par zone avec uniquement un dictionnaire.

Nous disposons de deux données :

1. Tableau des pays européens https://output.jsbin.com/lanasay.js
2. le dictionnaire des zones

Si nous disposons du tableau des pays européens, l'idée du filtre s'impose pour regrouper les pays par zone.

Nous allons filtrer les pays du tableau pour chaque zone, le tableau filtré est transformé en ensemble.

Voici le code pour définir les pays de la zone Ouest

```
const tabO = tabEurope.filter(function (country) {
        return mapEurope.get(country) == "O";
});
const europeO = new Set(tabO);
```

Code que l'on peut réécriture plus simplement :

```
const tabO = tabEurope.filter((country)=>mapEurope.get(country)=="O")
```

code

Nous nous fixons une nouvelle contrainte : nous ne disposons pas d'un tableau source contenant les pays. Ainsi, il nous est impossible de filtrer directement le tableau.

L'idée première est donc de filtrer directement le dictionnaire map.

Hélas, map ne dispose pas pour un dictionnaire des méthodes *filtre and Co*

Nous allons découvrir une solution élégante.

Nous utilisons l'opérateur de décomposition ...[3]

Rappels sur l'opérateur ... et la destructuration

L'opérateur ... transforme un Map en un tableau de tableau.

```
const mapEurope = new Map()
```

---

3 https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Syntaxe_d%C3%A9composition

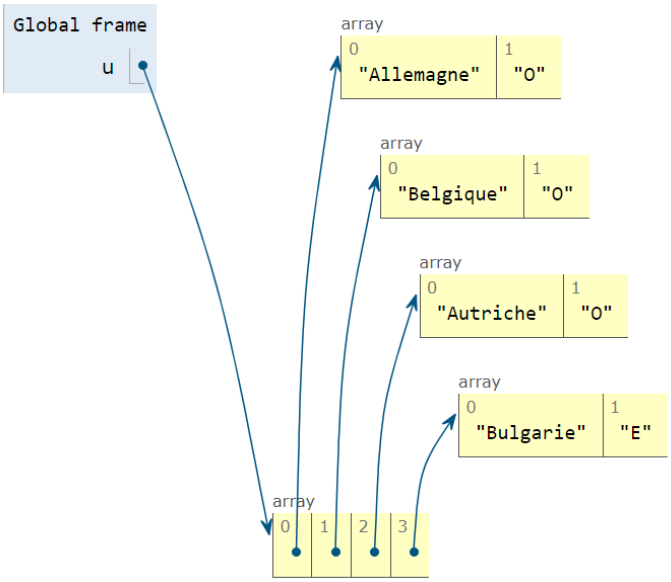
.set("Allemagne","O")

.set("Belgique","O")

.set("Autriche","O")

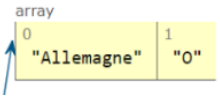
.set("Bulgarie","E");

const u = [...mapEurope];



On peut utiliser la destructuration sur les tableaux pour retrouver le pays et sa zone.

const tab = u[0]; // accès au premier tableau de u.



const [country, zone] = u[0];

Le tableau mapEurope peut être transformé à son tour en dictionnaire.

const u = [...mapEurope];

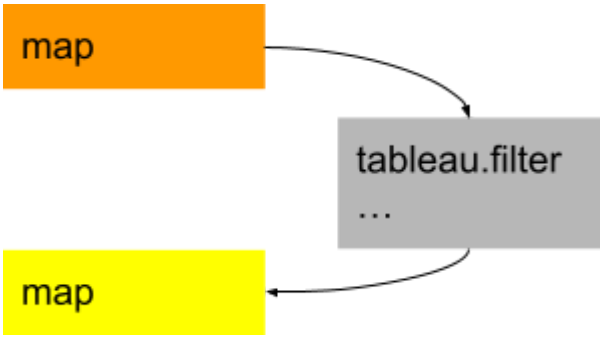
const mapEuropeBIS = new Map(u);

SOS->visualisez

Idée générale

Map ne disposant pas des méthodes *filtre and Co,* l'idée est de passer par un tableau intermédiaire, d'opérer les méthodes connues de filtre sur ce tableau puis de recréer un Map.

Voici une illustration du principe.



Voici le code pour regrouper les pays par zone si nous ne disposons uniquement d'un dictionnaire (Pays, Zone).

```
1.  let europeFindZone= new Map(
2.      [...mapEurope] .filter(function( [country, zone] ) {
3.        return zone == findZone;
4.      })
5.  );
```

SOScode

Lig. 2 : L'opérateur [...] va transformer mapEurope en un tableau. Il est de nouveau possible d'utiliser notre méthode filtre. Notez **la destructuration** avec [country, zone]

Lig. 3 : On teste si le pays est dans la zone à charger.

Lig. 1 : On recréé un Map.

Nous n'avons pas besoin ici d'informations sur le pays (country). On pourra écrire la destructuration ainsi [_, zone].

Dans une nouvelle version de code, nous utilisons l'objet ***this*** dont la référence est passée en dernier paramètre de la méthode filtre !

```
1.  const findZone = {zone : "O"};
2.
3.  let europeO= new Map(
4.      [...mapEurope]
5.      .filter(function([country, zone]) {
6.        return zone == this.zone;
7.      }, findZone)
8.  );
```

Lig.7 : le dernier paramètre permet de lier[4] ***this*** à l'objet passé en second argument. Ainsi ***this*** correspond dans la méthode filtre à l'objet findZone. Notez la puissance de cette astuce.

[4] On parle du "binding" en anglais

Continuons à améliorer notre code et posons nous la question :

⏩ Comment grouper "automatiquement" les pays par zone pour l'ensemble des zones ?

Pour le moment, nous déclarons dans une variable ***findZone*** la zone géographique à filter.

Nous voudrions nous passer de cette variable.

L'idée est de récupérer les zones directement dans le dictionnaire. C'est possible puisque les zones constituent les données du dictionnaire.

Nous savons comment récupérer les données d'un dictionnaire avec la méthode values(). Nous allons nous amuser à trouver d'autres idées.

Commençons par déterminer toutes les zones géographiques. Nous voulons extraire cette information du dictionnaire *mapEurope*.

forEach Map

La première idée est de faire une boucle sur le Map et d'ajouter les zones à un ensemble[5].

```
1. const zones = new Set();
2. function findZones(zone, country, map) {
3.     zones.add(zone);
4. };
5. mapEurope.forEach(findZones);
```

lig. 2 : notez l'ordre des arguments (value,key)

[5] On se rappelle qu'un ensemble a des valeurs uniques !

lig. 3 : on ajoute la zone à l'ensemble. Rien ne se passe si la zone existe déjà dans l'ensemble.

code

On peut également utiliser directement un itérateur sur les valeurs

```
1. const zones = new Set();
2. const it = mapEurope.values()[6] [7];
3.
4. for(let i of it){
5.   zones.add(i);
6. }
```

code

Zones est un ensemble. Ainsi pour pouvoir utiliser les méthodes filter and co, il convient de transformer l'ensemble en tableau.

Cette astuce est la même que celle appliquée au dictionnaire.

Nous allons transformer un tableau contenant les zones en un tableau regroupant pour chaque zone les pays de cette zone. Pour transformer un tableau, nous utilisons la méthode **map**.

map Tab

```
1. const tableau = [...zones];
2.
3. const mapToZone = function(zone){
4.   return new Map(
5.     [...mapEurope]
```

[6] it est un itérateur, ce n'est pas un tableau.
[7] mapEurope.keys() est également itérateur sur les 27 pays

```
6.      .filter(function([country, z]) {
7.        return z == zone;
8.      })
9.      )
10.}
11.
12.const allMap = tableau.map(mapToZone);
```

code

Lig. 1 : l'ensemble est transformer en tableau

Lig. 12: le tableau est transformé en un tableau de Map

Lig. 3 : la fonction de transformation renvoie un dictionnaire de tous les pays par zone.

Lig. 5 : l'opérateur [...] va transformer mapEurope en tableau.

Lig. 6 : il est de nouveau possible d'utiliser notre méthode filtre. Notez la destructuration avec [country, zone]

Lig. 7 : on teste si le pays est dans la zone.

On peut finalement destructurer le résultat.

const [europeO, europeE, europeS, europeN] = t.map(mapToZone);

Nous pouvons utiliser les fonctions =>

code

🔻 Attention : le nommage des zones dépend de l'ordre du mapEurope !

Si le map commence par un pays de la zone nord, il faudrait modifier le nommage des valeurs des zones.

Le code montre que la zone ne correspond plus aux pays, car nous avons modifier le premier objet du dictionnaire.

Il existe plusieurs solutions pour garantir l'affectation des zones aux noms correctes

On peut filtrer pour récupérer dans le map la valeur de la zone.

Voici comment on garantie une affectation correcte pour la zone "S"

```
1. const mapZones = t.map(mapToZone);
2.
3. const europeS = mapZones.filter(function(map){
4.   let [firstValue] = [...map];
5.   return firstValue[1] == "S";
6.
7. })
```

lig. 5 firstValue[1] est la zone commune à tous les pays dans le map. Elle sert de filtre par rapport à la zone recherchée.

Le code illustre le principe sur un dictionnaire réduit à cinq pays couvrant toutes les zones.

Hors programme[8]

voici une autre écriture du filtre, qui utilise le fait que la méthode values() renvoie un objet Iterator qui contient les valeurs de chacun des éléments contenu dans map.

```
1. const mapZones = t.map(mapToZone);
2.
3. const europeS = mapZones.filter(function(map){
```

[8] les itérateurs ne pas pas encore enseignés

```
4.    let zone = map.values().next().value
5.    return zone == "S";
6.  })
```

Lig. 4 : utilisation de la valeur de l'itérateur 1

code

sort tab

Autre idée, nous pouvons trier avec la destructuration les map

```
1.  const inc = function(mapA,mapB){
2.    let valA = mapA.values().next().value,
3.        valB = mapB.values().next().value;
4.
5.    return valA>valB?1:-1
6.  }
7.  const [europeE,europeN,europeO,europeS] = mapZones.sort(inc);
```

lig. 7 : les identifiant sont à déclarer par ordre alphabétique E,N,O,S

code

reduce tab

Nous ne pouvons utiliser la méthode reduce[9] sur le Map.

L'astuce devient classique : *"transformer le dictionnaire en un tableau."*

[9] slice n°101 du cours http://dupontes6.blogspot.com/p/7-array.html

```
1.  let tab = [...mapEurope];
2.  const zone = tab.reduce((a, [country, zone]) => {
3.    if (!a[zone]) a[zone] = [];
4.    a[zone].push(country);
5.    return a;
6.  }, {});
7.
```

Lig. 1 : on transforme le Map en tableau

lig. 2 : reduce, la valeur courante est destructurée.

lig. 3 : Ajout de la zone.

lig. 4 : Ajout du pays dans la zone.

lig. 6 : Nous initialisons reduce avec un objet vide {}. Le type retourné par reduce sera un objet.

SOS code

⬆️A partir d'un dictionnaire pays-Zone, on définit un dictionnaire zone-pays avec

```
1.  let tab = [...mapEurope];
2.
3.
4.  const zones = tab.reduce((a, [country, zone]) => {
5.      if (!a[zone]) a[zone] = [];
6.      a[zone].push(country);
7.      return a;
8.  }, {});
9.
10.
11. const mapZones = new Map(Object.entries(zones));
```

```
console.log(mapZone)
```

```
▼ Map(4) {"O" => Array(6), "E" => Array(7), "S" => Array(7), "N" => Array(7)}
  ▼ [[Entries]]
    ▶ 0: {"O" => Array(6)}
    ▶ 1: {"E" => Array(7)}
    ▶ 2: {"S" => Array(7)}
    ▶ 3: {"N" => Array(7)}
      size: (...)
    ▶ __proto__: Map
```

Affichage

Toutes les méthodes vues précédemment vont nous permettre de transformer nos valeurs avant leur affichage dans une grille.

Afficher tous les pays par zone

 index.html

```
<section class="europe"></section>
```

 style.css

```
.europe {
        display: grid;
        grid-template-columns: [N] 100px [O] 100px [E] 100px [S] 100px;
       /*   grid-auto-flow: row dense; */
        grid-auto-flow: column;
        grid-gap: 5px;
}
[data-cat="N"] {
        grid-column: N;
}
```

 script.js

```
async function get() {
  try {
```

```
    const response = await fetch("https://output.jsbin.com/lanasay.js");
    const countries = await response.json();
    initialize(countries);
  } catch (error) {
    console.log(error);
  }
}

get();

const mapEurope = new Map()
  .set("Allemagne", "O")
  .set("Belgique", "O")
…
  .set("Suède", "N");

function initialize(countries) {
  let sectionUI = document.body.querySelector(".europe");
  for (let i = 0; i < countries.length; i++) {
    const country = countries[i];
    let p = `<p data-cat=${mapEurope.get(country)}>${country}</p>`;
    sectionUI.insertAdjacentHTML("beforeEnd", p);
  }
}
```

SOS code

SOS code

⬆️ Imaginez que nous voulions afficher deux pays par régions !

Pas si simple !

Voici les grandes étapes résumées

Map	reduce	Object.entries(zones)
const mapEurope = new Map() .set("Allemagne","O") .set("Belgique","O") .set("Autriche","O") .set("Bulgarie","E") .set("Chypre","S") .set("Croatie","S") .set("Danemark","N") .set("Espagne","S") .set("Estonie","N")	{ 'O': ['Allemagne', 'Belgique'], 'E': ['Bulgarie', 'Hongrie'], 'S': ['Chypre', 'Croatie'], 'N': ['Danemark', 'Estonie'] }"	<section class="europe"> Allemagne Belgique Bulgarie Hongrie

🆘code

Autre code avec un tableau de pays

code

Les deux pays les plus peuplés

Tab	reduce	Object.entries(zones)

```
const Europe = [
  { nom: "Allemagne", zone: "O", population: 83 },
  { nom: "Belgique", zone: "O", population: 2 },
  { nom: "Autriche", zone: "O", population: 20 },
  { nom: "Bulgarie", zone: "E", population: 1 },
  { nom: "Chypre", zone: "S", population: 1 },
  { nom: "Croatie", zone: "S", population: 2 },
  { nom: "Danemark", zone: "N", population: 12 },
  { nom: "Espagne", zone: "S", population: 22 },
  { nom: "Estonie", zone: "N", population: 2 },
```

```
"{
  'O': {
    'nom': 'Allemagne',
    'zone': 'O',
    'population': 83
  },
  'E': {
    'nom': 'Pologne',
    'zone': 'E',
    'population': 45
  },
  'S': {
    'nom': 'Italie',
    'zone': 'S',
    'population': 55
  },
  'N': {
    'nom': 'Irlande',
    'zone': 'N',
    'population': 35
  }
}"
```

```
<section class="europe">
<h1 data-cat="O">O</h1>
<p data-cat="O">Allemagne: 83 </p>
<h1 data-cat="E">E</h1>
<p data-cat="E">Pologne: 45 </p>
<h1 data-cat="S">S</h1>
<p data-cat="S">Italie: 55 </p>
<h1 data-cat="N">N</h1>
<p data-cat="N">Irlande: 35 </p>
</section>
```

```
1.  const Europe = [
2.      { nom: "Allemagne", zone: "O", population: 83 },
3.      { nom: "Belgique", zone: "O", population: 2 },
4.  …
5.      { nom: "Suède", zone: "N", population: 31 }
6.  ];
7.
8.  const populatedCountry = Europe.reduce((a, country) => {
9.      if (!a[country.zone]) a[country.zone] = [{population:0},{population:0}];
10.
11.     const pays = a[country.zone];
12.     if (country.population > pays[0].population) {
13.       pays[0] = country;
14.     }
15.     else if ( country.population > pays[1].population) {
16.       pays[1] = country;
17.     }
18.
19.     return a;
20. }, {});
21.
22. // Interface graphique
```

```
23.
24. //Find Grid
25. let sectionUI = document.body.querySelector(".europe");
26.
27. //Pour chaque zone création de deux paragraphes
28. for (const [zone, countries] of Object.entries(populatedCountry)) {
29.   let titre = `<h1 data-cat=${zone}>${zone}</h1>`;
30.   sectionUI.insertAdjacentHTML("beforeEnd", titre);
31.
32.   for (let country of countries) {
33.     let p = `<p data-cat=${zone}>${country.nom}: ${country.population }</p>`;
34.     sectionUI.insertAdjacentHTML("beforeEnd", p);
35.   }
36. }
```

Notez lig. 9 l'importance de l'initialisation avec deux "pays" fictifs de population égale à Zéro.

Cette initialisation rend les tests plus simples.

code

EOF

Code

JS

```
const mapEurope = new Map()
.set("Allemagne","O")
.set("Belgique","O")
.set("Autriche","O")
.set("Bulgarie","E")
.set("Chypre","S")
.set("Croatie","S")
.set("Danemark","N")
.set("Espagne","S")
.set("Estonie","N")
.set("Finlande","N")
.set("France","O")
.set("Grèce","S")
.set("Hongrie","E")
.set("Irlande","N")
.set("Italie","S")
.set("Lettonie","N")
.set("Lituanie","N")
.set("Luxembourg","O")
.set("Malte","S")
.set("Pays-Bas","O")
.set("Pologne","E")
.set("Portugal","S")
.set("Tchéquie","E")
.set("Roumanie","E")
.set("Slovaquie","E")
.set("Slovénie","E")
.set("Suède","N");

let tab = [...mapEurope];

const zones = tab.reduce((a, [country, zone]) => {
  if (!a[zone]) a[zone] = [];
  if (a[zone].length < 2) a[zone].push(country);
  return a;
```

```
}, {});

let sectionUI = document.body.querySelector(".europe");

for (const [zone, countries] of Object.entries(zones)) {
  for (let country of countries) {
    let p = `<p data-cat=${zone}>${country}</p>`;
    sectionUI.insertAdjacentHTML("beforeEnd", p);
  }
}



/*
str = JSON.stringify(zones, null, 4); // (Optional) beautiful indented output.
console.log(str); // Logs output to dev tools console.
//alert(str);
*/
```

HTML

```
<section class="europe"></section>
```

CSS

```
.europe {
  display: grid;
  grid-auto-flow:column;
  grid-template-columns: [N] 100px [O] 100px [E] 100px [S] 100px;
```

```
/*   grid-auto-flow: row dense; */
  grid-gap: 5px;
}

[data-cat="N"] {
  grid-column: N;
}

[data-cat="O"] {
  grid-column: O;
}

[data-cat="E"] {
  grid-column: E;
}

[data-cat="S"] {
  grid-column: S;
}

body,
html {
  width: 100%;
  height: 100%;
}
```

Annexe

```
let pays = new
Set(["Allemagne","Belgique","Autriche","Bulgarie","Chypre","Croatie","Danemar
k","Espagne","Estonie","Finlande","France","Grèce","Hongrie","Irlande","Italie","Le
ttonie","Lituanie","Luxembourg","Malte","Pays-Bas","Pologne","Portugal","Tchéqu
ie","Roumanie","Slovaquie","Slovénie","Suède"]);


const mapt = new Map()
.set("Allemagne","O")
.set("Belgique","O")
.set("Autriche","O")
.set("Bulgarie","E")
.set("Chypre","S")
.set("Croatie","S")
.set("Danemark","N")
.set("Espagne","S")
.set("Estonie","N")
.set("Finlande","N")
.set("France","O")
.set("Grèce","S")
.set("Hongrie","E")
.set("Irlande","N")
.set("Italie","S")
.set("Lettonie","N")
.set("Lituanie","N")
```

```
.set("Luxembourg","O")
.set("Malte","S")
.set("Pays-Bas","O")
.set("Pologne","E")
.set("Portugal","S")
.set("Tchéquie","E")
.set("Roumanie","E")
.set("Slovaquie","E")
.set("Slovénie","E")
.set("Suède","N");

const europeE = new Set(
["Bulgarie","Hongrie","Pologne","Tchéquie","Roumanie","Slovaquie","Slovénie"]
);
const europeO = new Set(
["Allemagne","Belgique","Autriche","France","Luxembourg","Pays-Bas"]
);

const europeS = new Set(
["Chypre","Croatie","Espagne","Grèce","Italie","Malte","Portugal"]
);

const europeN = new Set(
["Estonie","Lettonie","Lituanie","Danemark","Finlande","Irlande","Suède"]
);
```

```
Autre code
const mapEurope = new Map()
  .set("Allemagne", {zone:"O"})
  .set("Belgique", {zone:"O"})
  .set("Autriche", {zone:"O"})
  .set("Bulgarie", {zone:"E"})
  .set("Chypre", {zone:"S"})
  .set("Croatie", {zone:"S"})
  .set("Danemark", {zone:"N"})
  .set("Espagne", {zone:"S"})
  .set("Estonie", {zone:"N"})
  .set("Finlande", {zone:"N"})
  .set("France", {zone:"O"})
  .set("Grèce", {zone:"S"})
  .set("Hongrie", {zone:"E"})
  .set("Irlande", {zone:"N"})
  .set("Italie", {zone:"S"})
  .set("Lettonie", {zone:"N"})
  .set("Lituanie", {zone:"N"})
  .set("Luxembourg", {zone:"O"})
  .set("Malte", {zone:"S"})
  .set("Pays-Bas", {zone:"O"})
  .set("Pologne", {zone:"E"})
  .set("Portugal", {zone:"S"})
  .set("Tchéquie", {zone:"E"})
  .set("Roumanie", {zone:"E"})
  .set("Slovaquie", {zone:"E"})
```

```
.set("Slovénie", {zone:"E"})
.set("Suède", {zone:"N"});
```