Objectifs

Nous allons nous intéresser à un littéral[1] de base : **le Tableau.**

 Le tableau est certainement le type de données le plus utilisé. Je vous passe la liste des algorithmes qui manipulent la structure de données du tableau.

littéral tableau

déclaration

Considérons un littéral tableau virus (on donne littéralement les éléments encadrés par des []) récupéré d'une base de données.

```
1. const virus =
   ["Arterivirus","Cytomegalovirus","H1N1","Lambda phage",
   "Myxovirusparotitidis","H3N2","Lentivirus","Leporipoxviru
   s","Astrovirus","Langur virus","Ateline herpesvirus
   group","Cytomégalovirus","Rabbit fibroma
   virus","Cytomegalovirus","Cytomegalovirus", "Vaccinia
   virus", "SUPERDUPONT", "Vacuolating
   virus","Coronavirus"];
```

Cette structure de données est manipulable par des algorithmes.

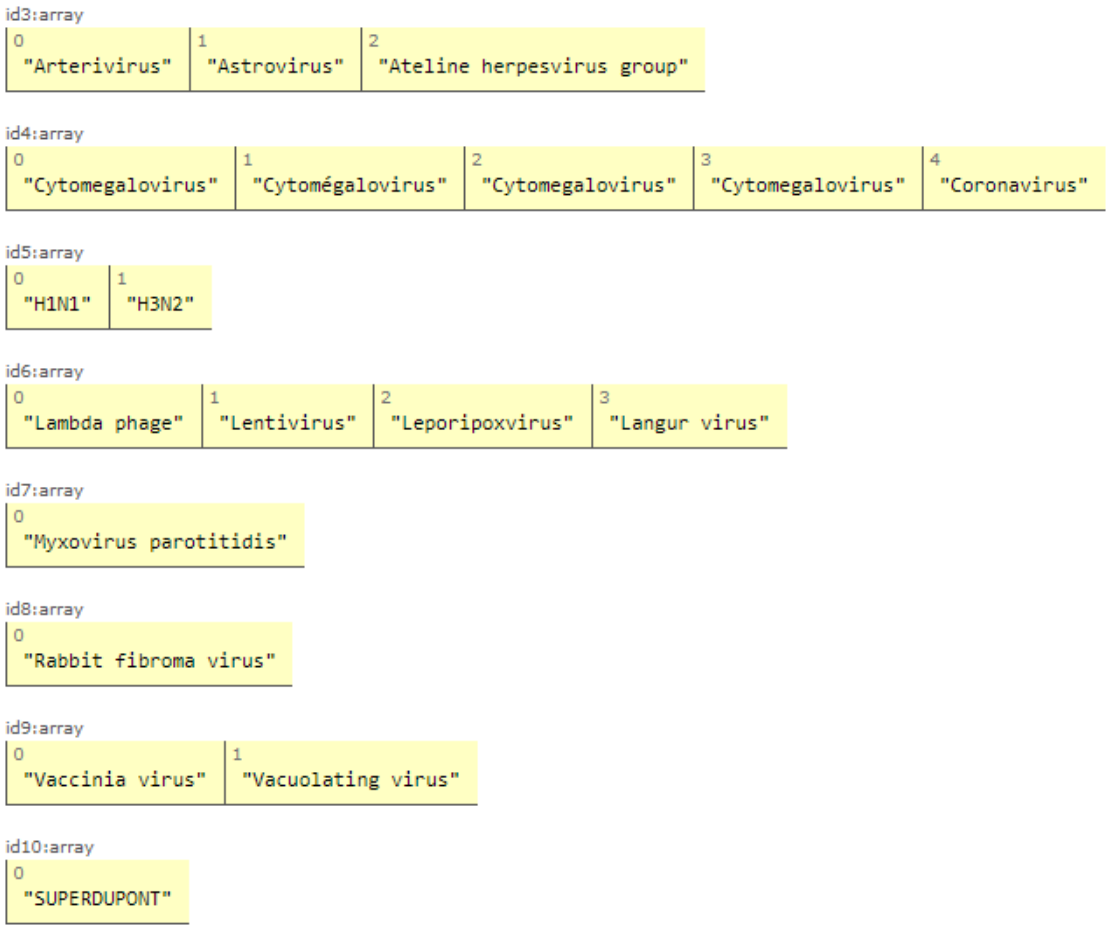
🥷Exemple d'utilisation

Voici un algorithme qui regroupe les virus dans un dictionnaire non ordonné.
En action

[1] Les littéraux sont utilisés pour représenter des valeurs en JavaScript. Un littéral tableau est une liste d'éléments d'un tableau, encadrée par des crochets [].

```
1.  const alphabetical = virus.reduce((a, x) => {
2.          if(!a[x[0]]) a[x[0]] = [];
3.          a[x[0]].push(x);
4.          return a; }, {}
5.          );
```

Résultats visuel



Cet algorithme montre l'utilisation de la méthode reduce. Cette méthode permet d'écrire un code très consis. Son utilisation peut paraître pour l'instant compliquée.

Dans cette introduction, nous nous contenterons de problèmes moins difficiles à conceptualiser.

Finalement, JavaScript empruntant la plupart des éléments de sa syntaxe à Java et C, nous devrions rapidement être à l'aise.

🪛Revenons à la notion de tableau.

Les tableaux

Un tableau est un objet.

Nous allons détailler un sous ensemble très réduit de propriétés et de méthodes disponibles pour tous les objets tableaux.

Propriétés

La propriété length d'un tableau donne la longueur du tableau. C'est à dire, le nombre d'éléments.

Indices

Les tableaux sont indexés à partir de zéro. Le premier élément d'un tableau a pour indice 0, et la position du dernier élément est donnée par length moins 1.

Dans l'exemple suivant, nous déclarons un littéral tableau(view). Ni la taille du tableau ni le type de ses éléments ne sont fixés. Cool !

```
const test = [1,2,"string"];
```

La valeur de test.length est égale à 3. C'est le nombre d'éléments contenus dans le tableau.

```
const test = [1,2,"string"];
```

Pour accéder au ième élément du tableau t, on utilise la notation t[i].

Le code suivant, modifie les valeurs aux indices 0 et 1 du tableau t. (lien)

1. const t = [-10,-20];
2. t[0] = 0;
3. t[1] = Math.abs(t[1]);

Les méthodes

Un tableau possède plusieurs méthodes incorporées pour exécuter des opérations de parcours et de modification.[2]

Pour utiliser une méthode sur un tableau, on utilise le . (point)

Voici un exemple (lien) :

test.js

1. const test = [1];
2.
3. test.push("new");
4. test.pop();
5. let value = test.pop();

Lig. 3, la méthode **push** ajoute à la fin du tableau un(des) élément(s) passé(s) en argument.

Lig. 4, la méthode **pop** retire un élément en fin du tableau.

[2] https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array#

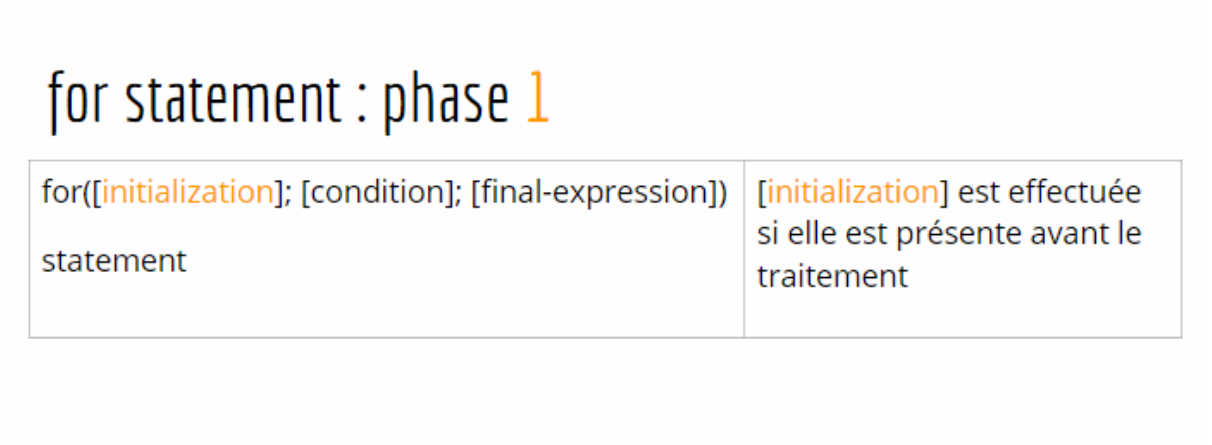
Lig. 5, la méthode **pop** retire un élément en fin du tableau que l'on affecte à une variable value.

Notez que test.push(v) est équivalent à test[test.length]=v

Les itérations

L'instruction **for**[3] définit une boucle composée de trois expressions optionnelles séparées par des points-virgules et encadrées entre des parenthèses qui sont suivies par un bloc d'instructions.

Les phases de l'itération avec le **for** sont rappelées dans l'animation suivante.



L'instruction **for** est donc tout à fait adaptée pour itérer sur les indices d'un tableau.

Une boucle **for** peut être interrompue en utilisant l'instruction **break.** On passe à l'itération suivante, si besoin, en utilisant le mot clé **continue.**

Remarque : il est facile d'écrire des "couillades". Trouvez au moins trois erreurs dans le code suivant (lien)[4].

Une autre forme d'itération existe `for..of`:, dans le cas, où vous n'avez pas besoin de l'indice (lien).

Modifiez le fichier test.js

[3] https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/for
[4] for (let i = tab.length-1 ; i >= 0 ; i--)

test.js

```
1. const villes = ["Evry", "Ovry", "Ivry"];
2.
3. for (let ville of villes) {
4.     console.log( ville );
5. }
```

Exercices[5]

Tous les exercices seront à ecrire avec pythontutor.

Vous pourrez communiquer le code sous forme de lien.

Dans la suite des questions, vous disposez uniquement des méthodes élémentaires (pop, push). **Donc pas de méthode sort(), filter() et autres !**

🆘Voici deux fonctions utiles que vous pouvez utiliser :

- La fonction random génère un nombre aléatoire (lien) :
- La méthode Array.from permet d'initialiser un tableau lien

1. EXO

Ni la taille du tableau ni le type de ses éléments ne sont fixés. Je signale que si fx est une fonction, fx() exécute cette fonction.

🪛Quel est le résultat de l'exécution de ce code ?

Modifiez le fichier test.js

test.js

```
1. let arr = ["Super", "Dupont"];
```

[5] **P**armi la liste des méthodes disponibles (plus de 40), nous n'utilisons qu'un ensemble très restreint ! L'objectif est donc de simuler le comportement de méthodes existantes.

```
2.
3.  arr.push(function() {
4.    console.log( arr );
5.  })
6.
7.  arr[2]()
```

2. EXO

🪛Créer un tableau de dimension Dim avec des nombres aléatoires compris entre [0-Dim][6].

L'utilisation de **for** est imposée.

3. EXO

🪛Renverser[7] les valeurs d'un tableau dans un nouveau tableau.

4. EXO

🪛Trouver le min (ou le max) d'un tableau.

Amusez-vous à changer de critère.

Par exemple, recherchez :

1⃣le premier nombre négatif,

2⃣les nombres pairs,

3⃣…

5. EXO

🪛Utiliser une seule instruction **for**, pour trouver la valeur min/max d'un tableau.

Mettez votre code dans le fichier source[8] (lien) pour vérifier votre résultat.

---

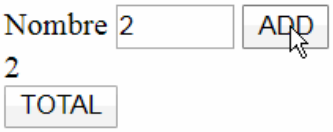
[6] Pour le fan club https://duponttd.blogspot.com/2016/09/creation-dun-tableau-es6-cool-code.html

[7] Vous avez compris, on ne doit pas utiliser la méthode reverse.

[8] Le code définit un tableau de valeurs aléatoires et compare les temps d'exécution.

6. EX0+

🥷Compléter le code JS pour additionner les valeurs saisies. jsfiddle (jsbin)



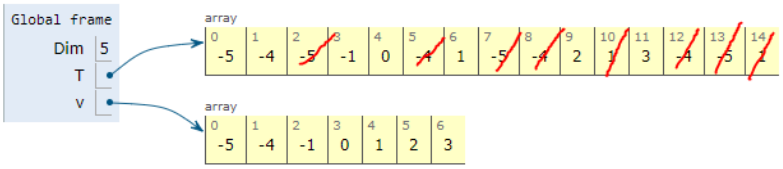
7. EX0

🪛Trouver le plus grand nombre dans un tableau à deux dimensions.

Mettez votre code dans ce fichier source pour vérifier votre résultat (lien)

8. EX0

🪛Extraire d'un tableau les valeurs uniques. (lien).



9. EX0

Voici les chiffres comparant deux programmes qui font la même chose.

```
denis.dupont@dd-HP MINGW6
$ node test.js
MonCode: 1496981.877ms
TonCode: 1126439.264ms
499604 versus 393133
```

🪛Trouvez un moyen simple de gagner des itérations pour l'algorithme suivant.

Compte tenu de la taille des tableaux, le code est à saisir dans Visual Studio code (on ne peut utiliser pythontutor).

Modifiez le fichier test.js

test.js

```
1. const Dim = 1000000,
2. A = Array.from({length: Dim},
3. () =>
   Math.floor(Math.random()>0.5?Math.random()*Dim:Math.random()*-Di
   m)),
4.
5. B = Array.from({length: Dim},
6. () =>
   Math.floor(Math.random()>0.5?Math.random()*Dim:Math.random()*-Di
   m));
7.
8. let t = [];
9. console.time("MonCode")
10.
11.for (let i = 0; i < A.length; i++) {
12.    let v = A[i];
13.    let f = false;
14.    for (let j = 0; j < B.length; j++) {
15.        if (B[j] == v) f = true;
16.    }
17.    if (f) t.push(v);
18.}
19.console.timeEnd("MonCode");
20.//------------
```

10. 🥷 EXO

Considérez deux tableaux G1 et G2. Trouvez

- les éléments communs aux deux tableaux
- les éléments de G1 pas dans G2
- Les éléments de G2 pas dans G1

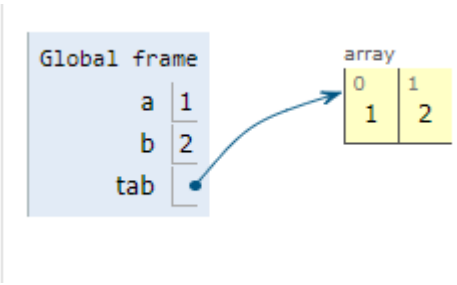
🆒Fin des exercices

👹Soyez très très attentif à la suite, cela parait simple, mais cela ne l'est pas.

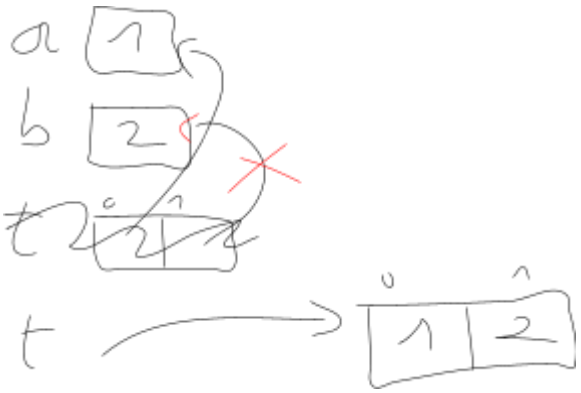
Soit le code suivant

```
1. let a = 1,
2.     b = 2,
3.     tab = [a,b];
```

Sa représentation est mémoire est donnée dans la figure suivante :

code	en mémoire (vérifier)
1. let a = 1, 2. b = 2, 3. tab = [a,b];	

J'ai pour le moins hésité sur le sens de la représentation



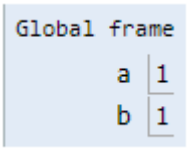
Rappels

les types primitifs

soit le code suivant

1. let a = 1,
2. b = 1;

Pour tous les types primitifs, la représentation est la suivante (lien) :



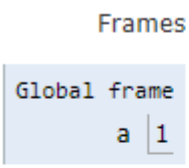
Ajoutons que la comparaison est réalisée par valeur. Deux valeurs sont strictement égales si elles ont la même valeur.

Un type primitif est dit immutable ; sa valeur ne change pas. AIE !

😳J'ai mis des années à comprendre ; dommage ! Vous avez 10 secondes pour comprendre.

Examiner le code suivant

1. let a = 1;
2. a = 1+3;



L'animation n'aide hélas en rien. Mais, voici deux hypothèques :

A. Le chiffre 1 se transforme par magie en 4 !

B. L'expression 1+3 est évaluée et retourne 4, puis l'expression a=4 est évaluée.

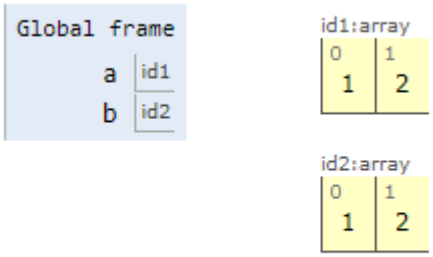
Les types non primitifs

Les valeurs non primitives sont des types de données mutables[9].

Examiner le code suivant

1. const a = [1,2];
2. const b = [1,2];

Pour les types non primitifs la représentation est la suivante (lien) :



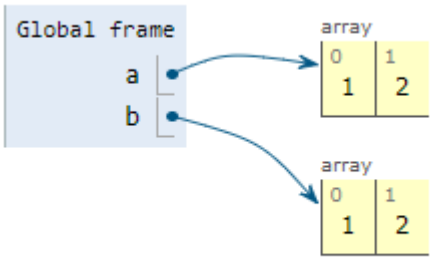
Dans la figure ci-dessus, on voit que

- a est une référence de valeur id1.
- b est une référence de valeur id2.

Ainsi a n'est pas égale à b.

Par contre, il n'est pas facile de trouver à quel objet a et b se réfèrent. On préfère la représentation avec les références explicites (lien)

[9] La valeur d'un objet peut être modifiée après sa création.



Par opposition au type immutables, on peut modifier un type mutable. C'est pourquoi dans les langages de programmation, on peut simplement modifier le contenu d'un tableau.

a[0] = 10;

Nous allons revenir en détail sur le terme de référence aux travers d'un certain nombre d'exemples.

Dessiner

Voici donc, une série de questions, vous devez dessiner l'état de la mémoire à l'issue de l'exécution du code.

code	en mémoire vérifier
1. const a = 1, 2. b = 2, 3. tab = [a,b];	

code	en mémoire vérifier
1. const a = 1, 2. b = [-10,-20]; 3. 4. b[0] = a; 5. b[1] = Math.abs(b[1]);	

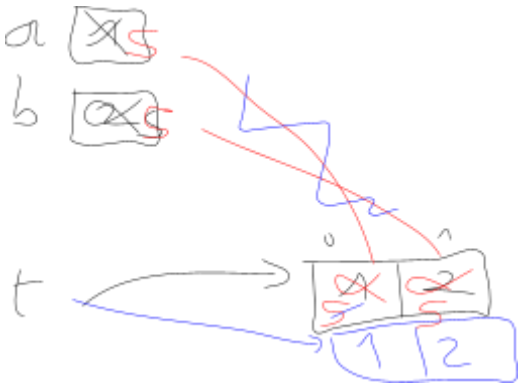
🪛Si vous avez tout compris dessiner l'état de la mémoire après exécution du code :

Modifiez le fichier test.js

test.js

```
1. const a = 1,
2.       b = 2,
3.       tab = [a,b];
4.
5. b = a = 5;
```

Comparez votre dessin avec le mien[10].

code	en mémoire vérifier
1. const a = 1, 2. b = 2, 3. tab = [a,b]; 4. 5. b = a = 5;	

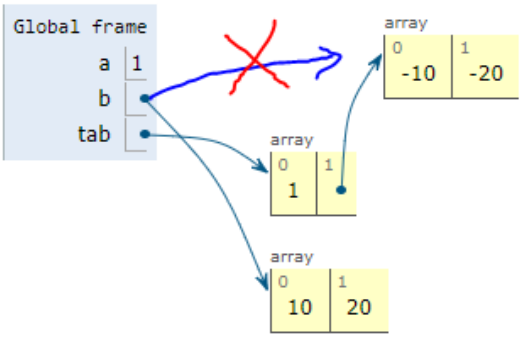
Dessinons maintenant l'état de la mémoire après exécution du code.

Modifiez le fichier test.js

test.js

```
1. let a = 1,
2.     b = [-10,-20];
3.
4. const tab = [a,b];
5.
6. b = [10, 20];
```

[10] J'ai encore un peu hésité sur la valeur de tab.

code	en mémoire vérifier
1. let a = 1, 2. b = [-10,-20]; 3. 4. const tab = [a,b]; 5. 6. b = [10, 20];	

Lig.4 tab[0] est une recopie de la valeur de a et tab[1] est une recopie de la référence du tableau référencé par b.

Lig.6 b va référencer un autre tableau. Notez bien que le tableau [-10,-20] reste visible car il est toujours référencé par tab[1].

⚠ Pouvez vous dessiner, le résultat après exécution du code ?

Modifiez le fichier test.js

test.js

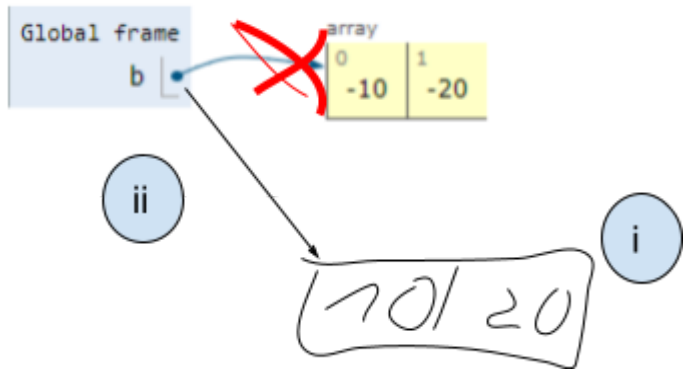
1. let b = [-10,-20];
2. b = [10, 20];

code	en mémoire vérifier
1. let b = [-10,-20]; 2. b = [10, 20];	

Reprenons étape par étape l'exécution du code dans pythontutor.

Dans la figure suivante, le passage de l'étape 2 à 3, est décomposé en deux étapes intermédiaires non visible dans le débogueur PythonTutor :

i. Création d'un tableau en mémoire
ii. b référence ce "nouveau tableau"



Il faut donc comprendre, que les valeurs du tableau [-10,-20] n'ont pas été modifiées pour devenir [10,20][11].

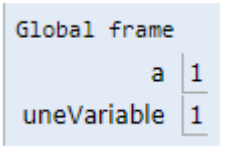
Il y a eu création d'un nouveau tableau et b devient sa référence.

Mais, que devient le tableau [-10,-20][12] ?

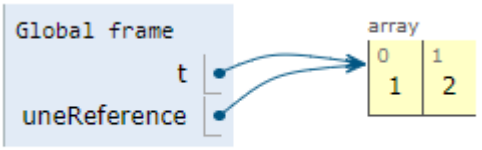
Résumé

L'affectation dépend du type de l'objet.

a est de type Number	recopie de la valeur

let a = 1;
 	
let uneVariable;
 	
uneVariable = a;
	

Ceci est vrai quelque soit l'objet primitif.

t est de type tableau	recopie de la référence
let t = [1,2] let uneReference; uneReference = t;	

[11] ce cas n'est pas sans rappeler celui déjà rencontré : les types primitifs

[12] Sa référence (b) est perdue, il n'est plus référençable (visible).

Ceci est vrai quelque soit l'objet non primitif.

1.