|  |
| --- |
| **JAVASCRIPT NIVEAU 2** |

|  |
| --- |
| **Var et Let** |

**Q1.** Regarde la vidéo ci-dessous et expliquez la différence entre les déclarations de variables Var let et const :

<https://www.youtube.com/watch?v=59YAPY2WYU0>

var est accessible dans la fonction et en dehors de la fonciton

Let est accessible dans le bloc et pas en dehors de la fonction

Un objet qui est déclaré dans une constante ne va pas être modifier dans une globalité en revanche on peut modifier chaque entrée dans l’objet

**Q2.** Quelle est la variable à choisir pour déclarer une variable d’incrémentation pour une boucle for ?

La variable let est la plus utile pour la boucle for

|  |
| --- |
| **Callback** |

Une fonction de rappel (aussi appelée callback en anglais) est une fonction passée dans une autre fonction en tant qu'argument, qui est ensuite invoquée à l'intérieur de la fonction externe pour accomplir une sorte de routine ou d'action.

**Q3.** Tester et commenter le programme ci-dessous.

La fonction départ lance la fonction dite « callback » qui a pour argument affichenom , qui va en premier temps demander le nom en prompt puis envoyer ce nom dans une autre fonction pour le traduire en texte avec l’ajout de salut et -\_-

|  |  |
| --- | --- |
| **script.js** | **index.html** |
| function afficheNom (nom){  alert('Salut ' + nom + ' -\_-')  }  function lireNom(callback){  votreNom=prompt('Entrez votre Nom');  callback(votreNom);  }  function Depart(){  lireNom(afficheNom);  } | <!DOCTYPE html>  <html>  <head>  <title>Javascript Test</title>  <script src="./script.js"></script>  </head>  <body>  <h1>Javascript Test</h1>  <p>Tester votre fonction : <button onclick="Depart()">Test</button></p>  </body></html> |

**Q4.** Regarder la vidéo suivante tester et expliquer le code synchronique et asynchrone. <https://www.youtube.com/watch?v=XKVgCCjz9EY&t=285s>

Le code synchronique suivera l’ordre d’éxécution même si il faut attendre un timer , tous le contraire du asynchrone qui outrepassera le timer et passera a la prochaine etape et reviendra a l’ancienne etape quand le timer sera terminer

Commenter le code ci-dessous et préciser pourquoi est-il asynchrone ensuite le mettre sous forme synchronisée en remplaçant console.log("BONJOUR") par un deuxième setTimeout .

Ce code est asynchrone a cause du timer qu’a l’étape 1

Ref « Résynchronisation avec le setTimeout.js » pour la deuxièmes partie de la question

|  |
| --- |
| // FONCTION DE CALLBACK ------------------  const snir =(ARG)=>{  console.log(ARG)  }  //----------------------------------------  setTimeout(snir,3000,"BTSSNIR"); //ETAPE 1 -- EXECUTE 2  console.log("BONJOUR"); //ETAPE2 --EXECUTE 1 |

|  |
| --- |
| **Objets Javascript** |

Tester le code en suivant :

|  |  |
| --- | --- |
| **script.js** | **Index.html** |
| **var personne = {**  **prenom: "John",**  **Nom: "Doe",**  **age: 50,**  **yeux: "blue"**  **};**  **function afficher(){**  **console.log(personne.Nom);**  **console.log(personne["prenom"]);**  **}** | **<!DOCTYPE html>**  **<html>**  **<head>**  **<title>Javascript Test</title>**  **<script src="./script.js"></script>**  **</head>**  **<body>**  **<h1>Javascript Test</h1>**  **<p>Tester votre fonction : <button onclick="afficher()">Test</button></p>**  **</body></html>** |

Aller sur le navigateur cliquez sur inspecter ensuite sur console pour visualiser le résultat.

|  |  |
| --- | --- |
|  |  |

Un objet javascript peut également prendre une fonction comme élément

tester le code suivant:

|  |
| --- |
| var personne = {  prenom: "Jules",  Nom: "Ferry",  age: 50,  informarion: function(){  return 'je suis ' +this.prenom + ' ' + this.Nom + ' ' + "j'ai " + this.age + ' ans';  }  };  function afficher(){  console.log(personne.Nom);  console.log(personne["prenom"]);  console.log(personne.informarion())  console.log(personne);  } |

**Q5.**Expliquer pourquoi on utilise this sur les éléments suivant this.prenom this.Nom this.age

Pour optimiser le code on prend les info qui sont dans le même object que la fonction

Voir la vidéo

<https://www.youtube.com/watch?v=gvicrj31JOM>

|  |
| --- |
| **Les propriétés Array(Foreach,map,slice,reduce)** |

Tester et commenter le code suivant :

|  |
| --- |
| var personnes =[  {  prenom: "Jules",  Nom: "Ferry",  age: 50,    informarion: function(){  return 'je suis ' +this.prenom + ' ' + this.Nom + ' ' + "j'ai " + this.age + ' ans';  }  },  {  prenom: "gerard",  Nom: "briaud",  age: 30,    informarion: function(){  return 'je suis ' +this.prenom + ' ' + this.Nom + ' ' + "j'ai " + this.age + ' ans';  }  }  ];  function afficher(){  personnes.forEach(element => console.log(element.informarion()));  } |

**Q5.**Expliquer la ligne de code element => console.log(element.informarion())

Voir :

C’est une fonction qui fera pour chaque x console.log(x.information) ici le x étant personne définit avant dans le personnes.foreach

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/Fonctions_fl%C3%A9ch%C3%A9es>

**Q6.**Refaire le code avec une fonction non fléchée

Ref sans la foncton flecher.html dans le github

**Q7.**Visualiser la vidéo suivante pour comprendre que array en javascript c’est un objet et remplir le tableau ci-dessous avec les différentes propriétés de l’objet

<https://www.youtube.com/watch?v=oigfaZ5ApsM>

|  |  |
| --- | --- |
| Propriété | Description |
| map | Créer un nouveau tableau avec le resutalt qu’on demande comme par exemple mettre tous un tableau en x2 |
| reduce |  |
| join | Changer tous les élément du tableau en 1 chaine de caractère |
| slice | Renvoie les objet du tableau mais qui n’est pas d’origine avec des élement en moins souhaite appeler portion la portion est définie par un indice de début et un indice de fin (exclus) |
| filter | Sers a séparé ce qu’on veut de ce qu’on veut pas en filtrant avec des conditions comme la taille ou une fonciton |
| Foreach | permet d'exécuter une fonction donnée sur chaque élément du tableau. |
| push | Rajoute a la fin du tableau l’élement souhaiter |
| length | Renvoie la longueur du tableau |
| find | renvoie la **valeur** du **premier élément trouvé**dans le tableau qui respecte la condition donnée par la fonction de test passée en argument. Sinon, la valeur [undefined](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/undefined) est renvoyée. |

Ajoutez le code ci-dessous à la fonction afficher.

|  |
| --- |
| const liste= personnes.map(x=> '<li>' + x.informarion()+'</li>');  console.log(liste); |

**Q7.**Que fait la fonction map sur le tableau ?.

Créer un tableau créant automatiquement les liste pour l’html avec la fonction informarion

**Q8.**Quelle est la forme de la variable liste ? . Tester le code ci-dessous à la place de

C’est du string

console.log(liste)

|  |
| --- |
| console.log(liste[0]);  console.log(liste[1]); |

Ajoutez le code ci-dessous à la place de

console.log(liste[0]);

console.log(liste[1]);

|  |
| --- |
| const html=liste.join('');  console.log(html); |

**Q8.**Quelle se passe-t-il avec la propriété join('') ?

Il transforme tous en 1 string avec pour espacement ‘ ‘

**Q9.**Pour afficher la variable html dans une page html utilisé le code ci-dessous et faire une conclusion.

|  |
| --- |
| const html=liste.join('');  console.log(html);  document.getElementById("affiche").innerHTML= '<ul>'+ html + '</ul>'; |

**Q10.**Ajouter deux personnes à votre tableau comme :

|  |
| --- |
| {  prenom: "Alexandre",  Nom: "Dumas",  age: 28,    informarion: function(){  return 'je suis ' +this.prenom + ' ' + this.Nom + ' ' + "j'ai " + this.age + ' ans';  }  },  {  prenom: "Remi",  Nom: "Gabin",  age: 25,    informarion: function(){  return 'je suis ' +this.prenom + ' ' + this.Nom + ' ' + "j'ai " + this.age + ' ans';  }  } |

**Q11** Ensuite, Faire un programme qui affiche sur la page html les personnes qui ont moins de 30 ans

Utilise la propriété filter()

Ref moins de 30ans.html

|  |
| --- |
|  |

**Q12** A la fin on souhaiterait afficher la moyennes d'âges de ce personnes en utilisant la fonction suivante:

|  |
| --- |
| function afficher(){  const age=personnes.map(x=>x=x.age);  console.log(age);  console.log(age.length)  const somme=age.reduce((moyenne,element)=>(moyenne+element))/age.length;  console.log(somme);} |

Pour comprendre la fonction reduce il faut visualiser la vidéo suivante:

<https://www.youtube.com/watch?v=g1C40tDP0Bk>

ensuite commenter le code précédent;

On peut également écrire le code précédent en une seul ligne comme suit :

|  |
| --- |
| const age=personnes.map(x=>x=x.age).reduce((moyenne,element)=>  (moyenne+element/personnes.length),0); |

**Q13** Maintenant afficher la moyenne d'âges sur une page HTML

|  |
| --- |
|  |

|  |
| --- |
| **Promise Javascript** |

L'objet **Promise** (pour « promesse ») est utilisé pour réaliser des traitements de façon asynchrone. Une promesse représente une valeur qui peut être disponible maintenant, dans le futur voire jamais.

Visualiser la vidéos suivantes pour comprendre le principe :

<https://www.youtube.com/watch?v=RvYYCGs45L4>

Visualiser et tester le code de la vidéo ci-dessous:

<https://www.youtube.com/watch?v=SSYt7C4sCbw>

|  |  |  |
| --- | --- | --- |
| Promesse 1 Partie 1 | Promesse 2 Partie 2 | Promesse 3 Partie 3 |
| //promesse Simple  const demarre=new Promise((resolve,reject)=>{  const isRunning =false;  if(isRunning == true){  resolve()  }else{  reject()  }  })  demarre.then(()=>{  console.log('good');  }).catch(()=>{  console.log("Oups, une erreur")  }) | // promesse avec argument dans resolve récupéré par then  const calcul = new Promise((resolve,reject)=>{  const result = 200\*120;    if(!result){  resolve(result)  }else{  reject(result)  }  })    calcul.then((result)=>{  console.log('good :'+result);  }).catch((result)=>{  console.log("Oups, une erreur:"+result)  }) | // promesse avec argument dans la fonction globale  const calcul = (num1,num2)=>{  return new Promise((resolve,reject)=>{  const result = num1\*num1;    if(!result){  resolve(result)  }else{  reject(result)  }  })  }      calcul(300,900).then((result)=>{  console.log('good :'+result);  }).catch((result)=>{  console.log("Oups, une erreur:"+result)  }) |

**Q14:**Modifier le code précédent promesse 3 en utilisant la méthode async await

**Q15:** Les promesses en javascript sont une solution pour les problèmes du code asynchrone.

Expliquez pourquoi !

|  |
| --- |
| **Une manière plus simple Async Await** |

Visualiser la vidéo ci-dessous et tester les deux méthodes de code

<https://www.youtube.com/watch?v=V_Kr9OSfDeU>

|  |  |
| --- | --- |
| **PROMISE** | **ASYNC AWAIT** |
| <script>  //-----------------Fonction1 pour faire une requete -----------------  function makeRequest(location){  return new Promise((resolve, reject)=>{  console.log(`Making Request to ${location}`)  if(location === 'Google'){  resolve('Google say hi')  }else {  reject(' We can only talk to Google')  }})}  //-----------------------------------------------  //----------------Fonction2 en cours de requete ----------------------  function processRequest(response){  return new Promise((resolve,reject)=>{  console.log('Processing response')  resolve(`Extra Information + ${response}`)  })}  // ----------------------------------------------------------  //------------------Faire une promesse --------------------------  makeRequest('Google').then(response=>{ //Appel fonction1->  console.log('Response Received')  return processRequest(response) //Appel fonction2 mettre la réponse de fonction 1 dans 2  }).then(processedResponse=>{ // faire un callback et afficher le resultat final  console.log(processedResponse)  }).catch(err=>{ //Exécute en cas d'érreur  console.log(err)})  //--------------------------------------------------------------  </script> | <script>  //-----------------Fonction1 pour faire une requete -----------------  function makeRequest(location){  return new Promise((resolve, reject)=>{  console.log(`Making Request to ${location}`)  if(location === 'Google'){  resolve('Google say hi')  }else {  reject(' We can only talk to Google')  }})}  //-----------------------------------------------  //----------------Fonction2 en cours de requete ----------------------  function processRequest(response){  return new Promise((resolve,reject)=>{  console.log('Processing response')  resolve(`Extra Information + ${response}`)  })}  // ----------------------------------------------------------  //-----------Async Await -----------------------  async function doWork(){  try{ // Essayer la requete  const response=await makeRequest('Google') // Appel fonction 1 et attendre la réponse  console.log('Response Received')  const processedResponse=await processRequest(response) // Appel fonction 2 attendre la réponse  console.log(processedResponse)  } catch(err){ //en cas d'erreur  console.log(err)}}    doWork()  </script> |

|  |
| --- |
| **Pratique !!!!!!!!!!!!!!! Fetch data** |