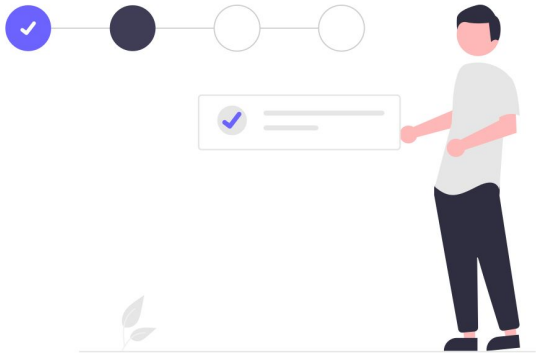
Javascript

Objectifs

1. Découvrir les **opérateurs conditionnels** en JS
2. Comprendre **le principe de portée**
3. **Manipuler** les opérateurs conditionnels

Découvrir les opérateurs conditionnels



Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Voici les différents opérateurs qu'il existe :

Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Voici les différents opérateurs qu'il existe :

- if… else (si...sinon)

Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Voici les différents opérateurs qu'il existe :

- if… else (si...sinon)
- else if (sinon si)

Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Voici les différents opérateurs qu'il existe :

- if… else (si...sinon)
- else if (sinon si)
- and / or (et / ou)

Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Voici les différents opérateurs qu'il existe :

- if… else (si...sinon)
- else if (sinon si)
- and / or (et / ou)
- switch...case (évaluer...attribuer)

Les opérateurs conditionnels

Un programme incapable de prendre une décision se révèle plutôt inutile. Par exemple, c'est grâce à une logique conditionnelle qu'il est possible de savoir si l'on est authentifié ou non sur un site web.

Voici les différents opérateurs qu'il existe :

- if… else (si...sinon)
- else if (sinon si)
- and / or (et / ou)
- switch...case (évaluer...attribuer)
- ternary (ternaires)

Les opérateurs conditionnels

If...else

L'instruction **if** permet de vérifier une condition (précisée entre les parenthèses). **Si la condition est évaluée comme vraie**, le code est alors exécuté.

Les opérateurs conditionnels

If...else

L'instruction **if** permet de vérifier une condition (précisée entre les parenthèses). **Si la condition est évaluée comme vraie**, le code est alors exécuté.

Nous pouvons ajouter une **autre instruction : else,** dans le cas où **la condition est évaluée comme étant fausse**.

Les opérateurs conditionnels

If...else

L'instruction **if** permet de vérifier une condition (précisée entre les parenthèses). **Si la condition est évaluée comme vraie**, le code est alors exécuté.

Nous pouvons ajouter une **autre instruction : else,** dans le cas où **la condition est évaluée comme étant fausse**.

```
if (name === "Paul") {
    console.log("Welcome, Paul");
} else {
    console.log("Go away!");
}
```

Les opérateurs conditionnels

If...else

L'instruction **if** permet de vérifier une condition (précisée entre les parenthèses). **Si la condition est évaluée comme vraie**, le code est alors exécuté.

Nous pouvons ajouter une **autre instruction : else,** dans le cas où **la condition est évaluée comme étant fausse**.

```
if (name === "Paul") {
    console.log("Welcome, Paul");
} else {
    console.log("Go away!");
}
```

Si le résultat est vrai, alors on affiche "**Welcome, Paul** " sinon on affiche "**Go away!**"

Les opérateurs conditionnels

Else if

Parfois, on peut avoir plus d'une condition… et avec la structure **if...else** uniquement, ce n'est pas vraiment possible.

Les opérateurs conditionnels

Else if

Parfois, on peut avoir plus d'une condition… et avec la structure **if...else** uniquement, ce n'est pas vraiment possible.

On va pouvoir ajouter des "embranchements" pour traiter plus de cas avec else if !

Les opérateurs conditionnels

Else if

Parfois, on peut avoir plus d'une condition… et avec la structure **if...else** uniquement, ce n'est pas vraiment possible.

On va pouvoir ajouter des "embranchements" pour traiter plus de cas avec else if !

```
if(user === "Paul"){
    console.log("Hello, Paul!");
}
else if(user === "John"){
    console.log("Hello, John!");
}
else if(user === "Ringo"){
    console.log("Hello, Ringo!");
}
else{
    console.log("Sorry, not today...");
}
```

Les opérateurs conditionnels

And et Or

Dans une condition, on peut aussi déterminer la véracité d'une **combinaison logique** de plusieurs **expressions** en utilisant les mots-clés **&&** ("and") et || ("or").

Les opérateurs conditionnels

And et Or

Dans une condition, on peut aussi déterminer la véracité d'une **combinaison logique** de plusieurs **expressions** en utilisant les mots-clés **&&** ("and") et || ("or").

```
const userName = "Paul";
const password = "secret";

if(userName === "Paul" || userName === "Bob"){
    console.log("Welcome dude!");
}

if(userName === "Paul" && password === "secret"){
    console.log("Welcome secret Paul!");
}
```

Les opérateurs conditionnels

```
const userName = "Paul";
const password = "secret";

if(userName === "Paul" || userName === "Bob"){
    console.log("Welcome dude!");
}

if(userName === "Paul" && password === "secret"){
    console.log("Welcome secret Paul!");
}
```

Dans ce code, on vérifie uniquement le nom de l'utilisateur dans le premier **if.** S'il est égal à Paul **OU** Bob, le code est exécuté.

Les opérateurs conditionnels

```
const userName = "Paul";
const password = "secret";

if(userName === "Paul" || userName === "Bob"){
    console.log("Welcome dude!");
}

if(userName === "Paul" && password === "secret"){
    console.log("Welcome secret Paul!");
}
```

Dans ce code, on vérifie uniquement le nom de l'utilisateur dans le premier **if.** S'il est égal à Paul **OU** Bob, le code est exécuté.

Dans le deuxième **if**, on vérifie que le nom vaut "Paul" **ET** que le mot de passe vaut "secret". Si les deux conditions sont remplies, le code est exécuté.

Les opérateurs conditionnels

```
const userName = "Paul";
const password = "secret";

if(userName === "Paul" || userName === "Bob"){
    console.log("Welcome dude!");
}

if(userName === "Paul" && password === "secret"){
    console.log("Welcome secret Paul!");
}
```

Dans ce code, on vérifie uniquement le nom de l'utilisateur dans le premier **if.** S'il est égal à Paul **OU** Bob, le code est exécuté.

Dans le deuxième **if**, on vérifie que le nom vaut "Paul" **ET** que le mot de passe vaut "secret". Si les deux conditions sont remplies, le code est exécuté.

Lorsque **&&** est utilisé, si **un des deux booléens vaut false**, tout sera évalué comme **false**.

Les opérateurs conditionnels

```
const userName = "Paul";
const password = "secret";

if(userName === "Paul" || userName === "Bob"){
    console.log("Welcome dude!");
}

if(userName === "Paul" && password === "secret"){
    console.log("Welcome secret Paul!");
}
```

Dans ce code, on vérifie uniquement le nom de l'utilisateur dans le premier **if.** S'il est égal à Paul **OU** Bob, le code est exécuté.

Dans le deuxième **if**, on vérifie que le nom vaut "Paul" **ET** que le mot de passe vaut "secret". Si les deux conditions sont remplies, le code est exécuté.

Lorsque **&&** est utilisé, si **un des deux booléens vaut false**, tout sera évalué comme **false**.

Lorsque **||** est utilisé, si **un des deux booléens vaut true**, tout sera évalué comme **true**.

Les opérateurs conditionnels

Booléen 1	and/or	Booléen 2	Result
true	&&	true	**true**
false	&&	false	**false**
true	&&	false	**false**
true	||	true	**true**
false	||	false	**false**
true	||	false	**true**

Les opérateurs conditionnels

Switch case

On peut également utiliser la structure **switch...case** pour évaluer une condition

Les opérateurs conditionnels

Switch case

On peut également utiliser la structure **switch...case** pour évaluer une condition

```
let userCountry = prompt("Where are you from");

switch(userCountry){
    case "France":
        console.log("Bonjour");
        break;
    case "England":
        console.log("Hello");
        break;
    case "Germany":
        console.log("Hallo");
        break;
    case "Italy":
        console.log("Ciao");
        break;
    case "Spain":
        console.log("Hola");
        break;
    default:
        console.log("Hey there");
        break;
}
```

Les opérateurs conditionnels

```
let userCountry = prompt("Where are you from");

switch(userCountry){
    case "France":
        console.log("Bonjour");
        break;
    case "England":
        console.log("Hello");
        break;
    case "Germany":
        console.log("Hallo");
        break;
    case "Italy":
        console.log("Ciao");
        break;
    case "Spain":
        console.log("Hola");
        break;
    default:
        console.log("Hey there");
        break;
}
```

On vérifie le pays d'origine de l'utilisateur et, en fonction de ce dernier, on affiche un message spécifique.

Les opérateurs conditionnels

```
let userCountry = prompt("Where are you from");

switch(userCountry){
    case "France":
        console.log("Bonjour");
        break;
    case "England":
        console.log("Hello");
        break;
    case "Germany":
        console.log("Hallo");
        break;
    case "Italy":
        console.log("Ciao");
        break;
    case "Spain":
        console.log("Hola");
        break;
    default:
        console.log("Hey there");
        break;
}
```

On vérifie le pays d'origine de l'utilisateur et, en fonction de ce dernier, on affiche un message spécifique.

Dans les parenthèses du **switch**, on met la valeur à comparer (avec l'opérateur **d'égalité stricte**) avec différents cas (**case**) possibles.

Les opérateurs conditionnels

```
let userCountry = prompt("Where are you from");

switch(userCountry){
    case "France":
        console.log("Bonjour");
        break;
    case "England":
        console.log("Hello");
        break;
    case "Germany":
        console.log("Hallo");
        break;
    case "Italy":
        console.log("Ciao");
        break;
    case "Spain":
        console.log("Hola");
        break;
    default:
        console.log("Hey there");
        break;
}
```

On vérifie le pays d'origine de l'utilisateur et, en fonction de ce dernier, on affiche un message spécifique.

Dans les parenthèses du **switch**, on met la valeur à comparer (avec l'opérateur **d'égalité stricte**) avec différents cas (**case**) possibles.

Le mot clé **default** sert à spécifier quoi faire si aucun des cas n'est égal à la valeur testée.

Les opérateurs conditionnels

Ternary

On peut simplifier l'écriture d'une condition en utilisant **l'opérateur ternaire**. Cet opérateur utilise **?** et **:**, respectivement équivalents à **if** et **else** :

Les opérateurs conditionnels

Ternary

On peut simplifier l'écriture d'une condition en utilisant **l'opérateur ternaire**. Cet opérateur utilise **?** et **:**, respectivement équivalents à **if** et **else** :

```
name === "Bob" ? console.log("Hello, Bob") : console.log("Go Away!");
```

Les valeurs truthy et falsy

Lorsque l’on écrit un bloc **if...else**, l'expression que écrite entre parenthèses est évaluée et transformée en booléen.

Les valeurs truthy et falsy

Lorsque l'on écrit un bloc **if...else**, l'expression que écrite entre parenthèses est évaluée et transformée en booléen.

Par exemple, l'expression **4 === 4** est **évaluée comme** ("transformée" en true). Si on place cette dernière dans les parenthèses d'un **if**, le code sous condition sera exécuté.

Les valeurs truthy et falsy

Lorsque l'on écrit un bloc **if...else**, l'expression que écrite entre parenthèses est évaluée et transformée en booléen.

Par exemple, l'expression **4 === 4** est **évaluée comme** ("transformée" en true). Si on place cette dernière dans les parenthèses d'un **if**, le code sous condition sera exécuté.

Mais que se passe-t-il si on met autre chose qu'un booléen à l'intérieur d'une condition ?

Les valeurs truthy et falsy

Lorsque l'on écrit un bloc **if...else**, l'expression que écrite entre parenthèses est évaluée et transformée en booléen.

Par exemple, l'expression **4 === 4** est **évaluée comme** ("transformée" en true). Si on place cette dernière dans les parenthèses d'un **if**, le code sous condition sera exécuté.

Mais que se passe-t-il si on met autre chose qu'un booléen à l'intérieur d'une condition ?

Par exemple :

Les valeurs truthy et falsy

Lorsque l'on écrit un bloc **if...else**, l'expression que écrite entre parenthèses est évaluée et transformée en booléen.

Par exemple, l'expression **4 === 4** est **évaluée comme** ("transformée" en true). Si on place cette dernière dans les parenthèses d'un **if**, le code sous condition sera exécuté.

Mais que se passe-t-il si on met autre chose qu'un booléen à l'intérieur d'une condition ?

Par exemple :

```
if (1) {
    console.log("What's going to happen here ? 🤷‍♀️");
}
```

Les valeurs truthy et falsy

Les valeurs dites "falsy" :

- false
- ""
- 0
- -0
- null
- undefined
- NaN

```
let myName = "";
if (myName) {
    console.log("Hello you!");
    // Le code écrit ici ne sera jamais lu puisque la condition
    // est évaluée comme fausse
} else {
    console.log("you don't have a name");
}
```

Toutes les autres valeurs sont "truthy" !

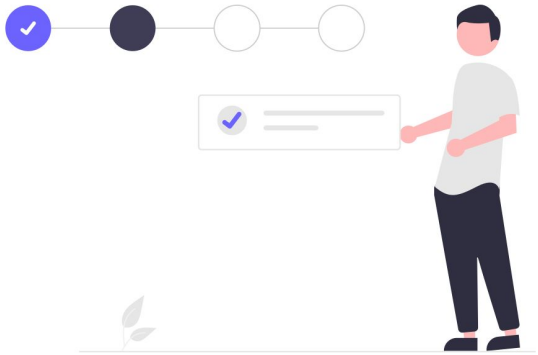
JS

Inversion logique

```
What's not true
is always false.
```

L'opérateur **!** permet d'**inverser** un booléen. Ainsi, **!false** est égal à **true** et **!true** est égal à **false**. Cet opérateur permet d'obtenir **l'opposé** d'une valeur.

Portée - contexte



En Javascript, dès que l'on écrit du code, le contexte est très important : **on ne peut pas utiliser une variable déclarée à l'intérieur d'une condition en dehors de cette dernière**.

En Javascript, dès que l'on écrit du code, le contexte est très important : **on ne peut pas utiliser une variable déclarée à l'intérieur d'une condition en dehors de cette dernière**.

Les accolades `{ }` définissent un contexte local.

En Javascript, dès que l'on écrit du code, le contexte est très important : **on ne peut pas utiliser une variable déclarée à l'intérieur d'une condition en dehors de cette dernière**.

Les accolades `{ }` définissent un contexte local.

```
const name = "Pierre";
const age = 18;

    if (name === "Pierre") {
        const city = "New York";
        if (city === "New York") {
            console.log("Welcome " + name + " in " + city);
        } else {
            console.log("Welcome " + name);
        }
        // fonctionne correctement à l'intérieur du contexte de la condition
    } else {
        console.log(age);
        // tu verras s'afficher l'age
       console.log(city)
       // tu verras une error 'reference error: city is not defined'
    }
```

Pratiquons !