HEIG-VD / INGÉNIERIE DES MÉDIAS / PROGRAMMATION WEB

FONCTIONS

```
function add(a, b) {
  return a + b;
}

const result = add(10, 5);
// → 15
```

Vous voudrez très certainement créer vos propres fonctions. La syntaxe de base est très simple.

Les fonctions retournent toujours quelque chose. Si aucune instruction `return` n'est donnée, le résultat est undefined.

```
let bird = "pigeon";

function watchBird() {
  let bird = "sparrow";
  console.log(bird);
}

watchBird(); // "sparrow"
console.log(bird);
// "pigeon"
```

Les variables `let` et `const` sont accessibles à l'intérieur de leur block. On appelle ceci le "block scope".

var

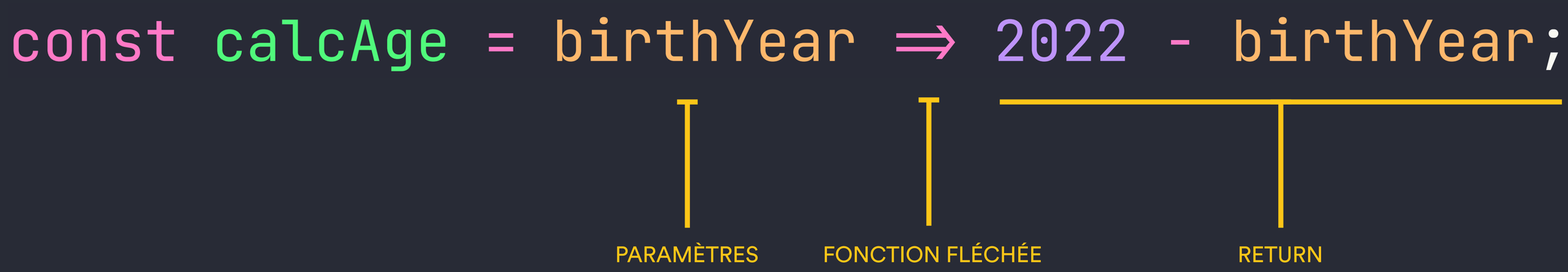
```
// Function declaration
function pow(a, n) {
  return a ** n;
}

// Function expression
const pow = function(a, n) {
  return a ** n;
}
```

En JS, les fonctions sont en réalité des objets. Ceci implique qu'on peut les stocker dans des variables. Une fonction peut donc prendre la forme d'une expression.

La seule différence entre ces deux techniques est le fait qu'on ne puisse pas appeler une expression de fonction avant l'avoir déclarée. (Hoisting)

HE VD IG


const calcAge = birthYear ⟹ 2022 - birthYear;
PARAMÈTRES
FONCTION FLÉCHÉE
RETURN

En JavaScript, les fonctions sont des citoyennes de première classe.

Ceci signifie que le langage permet de **passer des fonctions comme paramètres**, de les **recevoir en retour**, de les **attribuer à des variables** ou de **les stocker dans des structures complexes.**

HE IG VD

```
const flight = {
  airline: "Swiss",
  flight: "LX41",
  passengers: [],
  checkIn(passenger) {
    this.passengers.push(passenger);
 }
};

flight.checkIn("Simon");
flight.checkIn("Emilie");
console.log(flight.passengers);
// ["Simon", "Emilie"]
```

On peut déclarer des méthodes à l'intérieur d'un objet!

HE VD IG

```
const flight = {
  airline: "Swiss",
  flight: "LX41",
  passengers: [],
  checkIn(passenger) {
    this.passengers.push(passenger);
 }
};

flight.checkIn("Simon");
flight.checkIn("Emilie");
console.log(flight.passengers);
```

On peut déc
l'intérieur d'

```
  flight: "LX41",
  passengers: [],
  checkIn(passenger) {
    this.passengers.push(passer
}
;

Flight.checkIn("Simon");
```

```
  flight: "LX11",
  passengers: [],
  checkIn(passenger) {
    this.passengers.push
  }
};
```

```
    passengers: [],
    checkIn(passenge
      this.passenger
  }
};
```

```
checkIn(pa
    this.pas
}
```

this

👉 Une variable spéciale `this` est créé pour chaque contexte d'exécution (chaque fonction). Sa valeur pointe vers le "propriétaire" de la fonction où `this` est utilisé.

☝️ `this` n'est pas statique. Sa valeur dépend de la façon dont la fonction a été appelée et n'est qu'assignée que lors de l'exécution.

HE VD IG

Method 👉 `this = <Objet qui contient la méthode>`

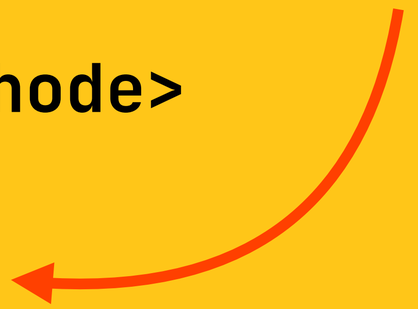
Method 👉 `this = <Objet qui contient la méthode>`

Appel de fonction standard 👉 `this = undefined`

UNIQUEMENT AVEC
“USE STRICT”!!!!

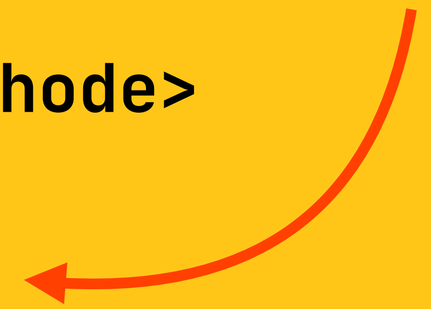
Method 👉 `this = <Objet qui contient la méthode>`

Appel de fonction standard 👉 `this = undefined`



HE VD
IG

UNIQUEMENT AVEC
“USE STRICT”!!!!



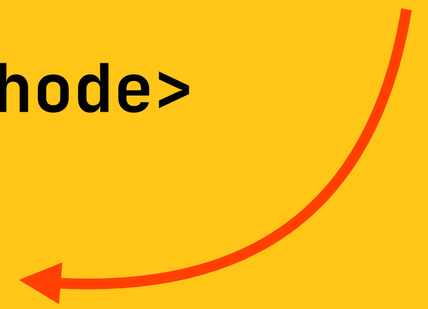
Method 👉 `this = <Objet qui contient la méthode>`

Appel de fonction standard 👉 `this = undefined`

Fonction fléchées 👉 `this = <this de la fonction parent>`

HE VD
IG

UNIQUEMENT AVEC
“USE STRICT”!!!!



Method 👉 `this = <Objet qui contient la méthode>`

Appel de fonction standard 👉 `this = undefined`

Fonction fléchées 👉 `this = <this de la fonction parente>`

(Event listener 👉 `this = <Element DOM attaché>`)

HE VD IG

```
const flight = {
  airline: "Swiss",
  flight: "LX41",
  passengers: [],
  checkIn(passenger) {
    this.passengers.push(passenger);
 }
};

flight.checkIn("Simon");
flight.checkIn("Emilie");
console.log(flight.passengers);
// ["Simon", "Emilie"]
```

À l'intérieur de vos méthodes, utilisez systématiquement `this` pour vous référer à des propriétés de l'objet en question.

Ceci vous permettra d'être plus abstrait lorsque vous souhaiterez générer des objets (avec un constructeur, par exemple).

HEIG VD

En JavaScript, les fonctions sont des citoyennes de première classe.

Ceci signifie que le langage permet de **passer des fonctions comme paramètres**, de les **recevoir en retour**, de les **attribuer à des variables** ou de **les stocker dans des structures complexes.**

HE IG VD