Solution

```
function fonctionUn(unTruc){
    console.log(33+unTruc);
}
fonctionUn(7);
```

💡 On peut aussi prévoir des fonctions nécessitant plusieurs paramètres comme ceci

```
function fonctionDeux(unTruc,unBidule){
    console.log(unBidule+unTruc);
}
fonctionDeux(10,88);
```

Return

Les fonctions gèrent également le concept de return, c'est-à-dire que nous pouvons écrire des fonctions qui vont retourner quelque chose (une variable, un résultat, etc..). Dans les exemples précédents, si on analyse bien, nos fonctions font deux choses techniquement : un calcul ET l'affichage du résultat de ce calcul, mais admettons, nous voulons garder un code clair et précis, on veut une fonction qui fait Uniquement du calcul, l'affichage du résultat sera géré ailleurs dans le programme.
Il faut donc adapter notre fonction pour qu'elle retourne un résultat que l'on stockera dans une variable.

Exemple avec une fonction qui a pour but de soustraire 2 nombres :

```
//! Dans certains cas une fonction doit pouvoir retourner quelquechose
//! le résultat d'un calcul par exemple
//! Ci-dessous on fait une fonction de calcul, notre fonction ne fait que ca
//! Elle se charge JUSTE de faire un calcul
//! L'affichage du résultat se fera en dehors de la fonction
function calculReturn(unNombre, unAutreNombre){
    return unNombre + unAutreNombre
}
//! Ici le calcul qui est return par la fonction est stocké dans une variable
//! resultat
let resultat = calculReturn(22,99);
console.log(resultat);
// ou executer la fonction quand on a besoin
console.log('Le résultat : ', calculReturn(22,99));
```

Paramètre par défaut

Une bonne pratique lorsque l'on écrit des fonctions (surtout en travail collaboratif), consiste à renseigner un paramètre par défaut dans le cas où on oublie de renseigner un paramètre quand on exécute une fonction.

```
//** Bonne Pratique : paramètre par défaut
function fonctionAvecParametre(num=0){
    console.log(22+num);
}
```

Scope

⚠️ Quand on code des fonctions (quand les scripts se complexifient), il est nécessaire de prendre en compte la notion de scope soit la portée des variables.
Une fonction va pouvoir utiliser des variables déclarées globalement mais le programme global ne peut pas utiliser une variable déclarée dans une fonction.

```
// ? La notion de scope (la portée d'une variable)
// ? Dans l'exemple ci-dessous on a 2 fois la même variable testScope1 qui est déclarée
?????
// ? En fait même si elles ont le même nom ce ne sont pas les même espaces mémoires qui
sont alloués
// ? let testScope1 = 99; est dans le scope global de notre programme
// ? let testScope1 = 12; est dans le scope de la fonction
let testScope1 = 99;
function maFonctionTestScope(){
    let testScope1 = 12;
    console.log('scope de la fonction :',testScope1);
};
maFonctionTestScope();
console.log('scope hors de la fonction :',testScope1);
```

Exercice : Quiz 1 Trouver le bug

```
//TODO : Pourquoi ca beug ?
function buggyFunction() {
    let wtf = 9;
    console.log(wtf);
};
buggyFunction();
console.log(wtf);
```

```
▸Uncaught ReferenceError: wtf is not defined        app.js:71
    at app.js:71:13
```

Exercice : Quiz 2 Trouver le bug

```
//TODO : Pourquoi ca beug / Pourquoi ca marche pas ?
let something = 44;
function functionBugParent() {
    let something = 9;
    console.log(something);
    console.log(lesNews);


    function functionBugEnfant() {
        let lesNews = `il est 99h67`;
    }
};
functionBugParent();
console.log(something);
```

```
▶Uncaught ReferenceError: lesNews is not defined    app.js:79
    at functionBugParent (app.js:79:17)
    at app.js:86:1
```

Exercice : Moyenne

```
//! EXO 5.2 : La moyenne de 2 notes
//TODO: Créer une fonction qui calcule la moyenne de 2 notes
//TODO: Afficher le résultat en console
```

Solution

```
let noteSport = 8;
let notePhilo = 2;
let laMoyenne = moyenne2notes(notePhilo,noteSport);
// On peut executer la f° AVANT de la définir (pas d'ordre pour décrire les fonctions)
function moyenne2notes(a,b){
    return (a+b)/2;
};
console.log('La moyenne des 2 notes : ',laMoyenne);
```

https://github.com/jefff404/cours-js/tree/6-functions

Opérateurs de comparaison / condition ternaire

Opérateurs de comparaison

Autre concept qui sera surtout utilisé pour les conditions, ce sont les opérateurs de comparaison cela renvoi un booléen (true ou false).

```
/**
 * ************************************
 * 7- Les opérateurs
 * ************************************
 */
//! Les booléens : 2 états possibles TRUE ou FALSE (vrai ou faux)
let a = 11;
let b = 99;
console.log("variable a:",a);
console.log("variable b:",b);
//! avec == on demande si a est égal à b
console.log("c'est égal ? :",a == b);
//!pour vérifier si a est différent de b on utilise !=
console.log("C'est inégal ? :",a != b);
//! Ensuite on retrouve les même opérateurs qu'en Mathématique
//! ici on demande si a est strictement suppérieur à b
console.log("Strictement suppérieur ? :",a > b);
//! ici on demande si a est strictement inférieur à b
console.log("Strictement inférieur ? :",a < b);
//! ici on demande si a est inférieur ou égal à b
console.log("Inférieur ou égal ? :",a <= b);
//! ici on demande si a est suppérieur ou égal à b
console.log("suppérieur ou égal ?:",a >= b);
//? Attention : de base JS ne prend pas en compte le typage des variables :
//? ci dessous le nombre 2 est égal au caractère "2" 🤔
console.log("le chiffre 2 = \"2\"?:",2 == "2");
//! Pour prendre en compte le type des donnée que l'on compare, on utilise l'opérateur
===
//! c'est l'égalité stricte
console.log("égalité stricte ?:",2 === "2");
//! il y a aussi l'inégalité stricte avec l'opérateur !==
console.log("inégalité stricte ?:",2 !== "2");
```

à noter la différence pour vérifier une égalité entre l'opérateur == et ===, le triple égale va permettre de vérifier aussi si les variables comparées sont du même type.

Condition / opérateur ternaire

Une syntaxe pour écrire des conditions simples qui renvoient quelque chose ou quelque chose d'autre si une condition est remplie ou non. Grâce à l'opérateur « ? »
Avant le ? on décrit notre condition, après le ? nous avons ce qui est return quand la condition est remplie puis « : » et ce qui est return quand la condition n'est pas remplie

```
//!-------CONDITIONS TERNAIRES-------
// ? on combine un opérateur de comparaison et l'opérateur ? pour établir une condition qui return une chose ou une autre chose
// ? cela permet de faire une condition if (simple) avec une syntaxe racourcie
let whatIsYourAge = 6;
console.log(whatIsYourAge >18 ? '🍹':'👮');
// Astuce pour check si une variable est définie (si ya QQchose dans son espace mémoire)
let userPremium;
// On check si une variable est définie la condition c'est juste uneVariable ?
console.log(userPremium?'OK 👍':'Not OK 👺');
// ↑ c'est l'équivalent de ↓
console.log(userPremium ==true?'OK 👍':'Not OK 👺');
// on doit lui assigner QQCHOSE
userPremium = 'YES';
console.log(userPremium?'OK 👍':'Not OK 👺');
```

On peut aussi combiner plusieurs conditions avec les opérateurs
|| (une condition OU une autre condition), && (une condition ET une autre condition)

```
// ? On peut utiliser des operateur aussi pour combiner des conditions && (pour ET) || (pour OU)
console.log(3==3&&3<4);
let typeUtilisateur = 'Extra';
console.log(typeUtilisateur == 'Extra' || typeUtilisateur == 'Premium');
```

Condition IF ELSE

Avec if, nous allons pouvoir exécuter du code seulement si une condition est remplie, on peut combiner if avec else qui correspond à SINON
Dans l'exemple ci-dessous nous avons une fonction qui prend un nombre en paramètre et à l'intérieur de cette fonction nous avons plusieurs conditions :
SI le nombre est supérieur ou = à 30 alors on return une phrase
SINON SI le nombre est inférieur à 10 alors on return une autre phrase
SINON on return une troisième phrase

```
//!--------CONDITION avec IF ELSE--------
// ? Avec if on va pouvoir créer un bloc de code qui s'exécute si une condition est remplie
function calculTableResto(nombreDeReservation) {
    if (nombreDeReservation>=30){
        return 'il nous reste pas beaucoup de tables, ca serait pour combien de personnes ?';
    }
    else if(nombreDeReservation<10){
        return 'Il nous reste une table'
    }…
    else{
        return 'On est fermé !'
    }
};
console.log(calculTableResto(50));
```

Exercice : If Else

```
//! EXO 7 - IF ELSE
// TODO: Créer une fonction qui reçoit un tableau de 3 notes et qui calcule une moyenne entre ces 3 notes
// TODO: Dans cette f°, SI la moyenne est suppérieur ou égale à 15 on renvoi une string (très Bien)
// TODO: Dans cette f°, SINON SI la moyenne est suppérieur ou égale à 10 on renvoi une string (assez Bien)
// TODO: Dans cette f°, SINON renvoi une string (refus)
```

Solution

```
function msgMentionBacOfficiel(tabNotes) {
    let moyenneCalc = (tabNotes[0]+tabNotes[1]+tabNotes[2])/tabNotes.length;
    console.log('la Moyenne au Bac : ',moyenneCalc);
    if (moyenneCalc>=16) {
        return "Tu as Gagné !"
    } else if (moyenneCalc >=10 && moyenneCalc<16) {
        return 'Assez Bien'
    } else {
        return 'YO T NUL GRO'
    }
};
console.log(msgMentionBacOfficiel([13,6,3]));
```

Objets

Autre type de variable utile pour stocker dans une variable plusieurs information, les objets avec la syntaxe en accolades { }, assez similaires aux tableaux (mais les objet n'ont pas de système d'indexation), c'est à nous de définir les propriétés (les clé) et leur assigner avec « : » une valeur.

```
// ? syntaxe  { uneProprieté:uneValeur }
// ? dans un objet on assigne avec : plutot qu'avec =
let user = {
    id:3657826,
    'name':'Seagal',
    firstname:'Steven',
    badges:['🎥','👮','🎸','🥋','🎤']
};
console.log(user);
```

On peut accéder aux propriétés d'un objet avec la notation en point

```
console.log(user.name);
console.log(user.id);
```

Ou avec la notation en tableau associatif

```
console.log(user['id']);
```

Pour ajouter une propriété on fait une assignation de valeur (si la propriété existe sa valeur est écrasée par la nouvelle, sinon cela créer la propriété)

On peut également effacer une propriété d'un objet avec delete

```
// ? On peut supprimer une propriété
delete user.badges;
```

```
// ? On peut ajouter simplement des propriétés dans un objet avec une assignation de valeur
// ? Si on assigne à une propriété déjà existante cela écrase la valeur
// ? Mais Si on assigne à une propriété non existante cela créee la propriété
user.dps = 9999;
```

hasOwnProperty

JS propose plusieurs fonctions natives utilisables sur des objets notamment. hasOwnProperty(), qui renvoi true ou false pour vérifier si la propriété d'un objet existe.

Ci-dessous on utilise la fonction dans un console.log() directement.

```
// ? une f° native de JS poour connaitre les propriétés d'un objet, hasOwnProperty()
let menuDuJour={
    entree:"Bassine d'Aioli",
    plat:"Rat Adulte",
    dessert:'île Fidji'
};
console.log(menuDuJour);
console.log(menuDuJour.hasOwnProperty('entree'));
```

```
true                                                    app.js:31
```

Exercice : Objects

```
// ! EXO 8 OBJECTS
// TODO : Faire l'exo avec le User et les passions en mode objet
```

Solution

```
// TODO : Faire l'exo avec les passions en mode objet
let nameUser = 'Dong Rodrigue';
let ageUser = 65;
let objetUser = {
    'nom' : nameUser,
    'age' : ageUser,
    'passions': {
        passion1:'Le Drift',
        passion2:'Jonquilles'
    }
};
console.log('objet de utilisateur : ',objetUser);
console.log(objetUser.nom);
console.log(objetUser['passions']);// c le taleau passions
console.log(objetUser.passions.passion2);

objetUser.name = 111;
objetUser.age = 'DonDiegoDelavega';
objetUser.passions.passion2 = 'Le Cinéma';
```

Comme pour les tableaux, dans un objet les propriétés peuvent être réassignées à une valeur

https://github.com/jefff404/cours-js/tree/9-objets