LANGAGE JAVASCRIPT: LES FONCTIONS

UTILITÉ

Les fonctions permettent de décomposer un problème en sous-problème. Ses sous-problèmes en termes de vocabulaire s'appellent des fonctions. Plusieurs intérêts:

- Écrire un code limité à une page.
- Réutilisation des fonctions déjà existante.
- Fonction propriétaire dont le code n'est pas lisible : Attention : ce n'est pas le cas en JavaScript.
- Éviter de répéter plusieurs morceaux de code identiques: le programme devient modulaire

Exemple:

```
let uneVal=0, puissance=0;
let i=0, nbPuissances=0;

uneVal = prompt("Donnez un nombre: ");
nbPuissances = prompt("Combien de puissances successives souhaitez-vous ? ");

//calcul des nbPuissances puissances successives de uneVal
puissance = 1;
for (let i = 1; i <= nbPuissances; i++) {
    puissance = puissance * uneVal;
}

document.write(uneVal  +"<sup>" +nbPuissances +"</sup>=" +  puissance);
```

$2^6=64$

Ce programme peut se remplacer par ce programme plus simple (4lignes remplacées par une seul) :

```
let uneVal=0, puissance=0;
let i=0, nbPuissances=0;

uneVal = prompt("Donnez un nombre: ");
nbPuissances = prompt("Combien de puissances successives souhaitez-vous ? ");

//calcul des nbPuissances puissances successives de uneVal
puissance = calculPuissance(uneVal,nbPuissances);

document.write(uneVal  +"<sup>" +nbPuissances +"</sup>=" +  puissance);
```

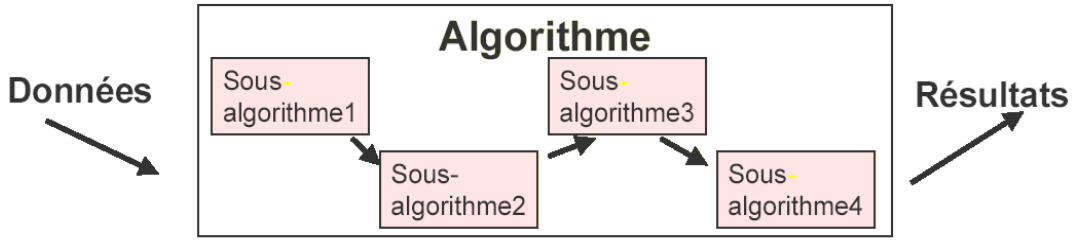
$2^6=64$

Ici **calculPuissance**() est une fonction (réutilisable à volonté) définie par ce code :

```
function calculPuissance(a,n) {
   let puissance = 1;
   for (let i = 1; i <= n; i++) {
       puissance = puissance * a;
   }
   return puissance;
}
```

GÉNÉRALITÉ

Avec les fonctions, les problèmes sont représentés et pensées de la façon suivante :



Une fonction est un ensemble **indépendant** d'instructions écrites pour accomplir une tâche précise.

Indépendant signifie que la fonction est réutilisable dans n'importe quel programme, sans apporter aucun changement au code initiale de la fonction.

2 SITUATIONS DE TRAVAIL :

- **On utilise une fonction existante** : l'utilisateur doit pouvoir faire abstraction du contenu de l'objet (boites noires) ou fonction. Seuls l'interface (entrées et sorties) et le rôle nous intéressent. Les questions à se poser sont :
  - Que fait la fonction ?
  - Comment l'utilise-t-on ? ➔ Il faut se référer au mode d'emploi: la signature de la fonction.

  Exemple : utiliser une fonction **convert_avi_to_mpeg()**. Le contenu, c'est-à-dire comment est codé la fonction n'est pas important ici. Notre vision est celle d'une boîte noire. Ce qui est important c'est comment utiliser la fonction.

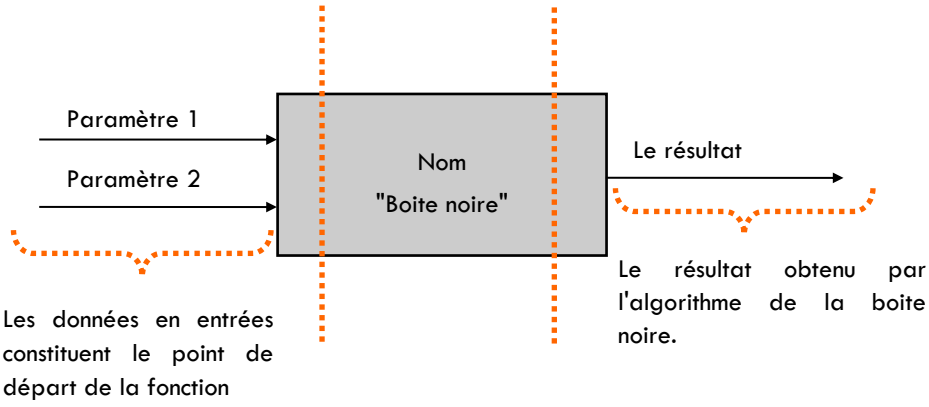
  Nous utilisons depuis le début de ce cours des fonctions existantes (**parseInt, ParseFloat, prompt, alert, confirm, isNaN,** etc…) sans connaitre le code associé à ses fonctions. Ce qui est important, c'est de savoir que ça fonctionne et comment ça marche
- **On développe une nouvelle fonction (conception)** : le programmeur doit faire abstraction du contexte dans lequel la fonction sera utilisée. Une fonction bien conçue est une fonction que l'on peut réutiliser d'un programme à l'autre sans que l'on soit obligé de modifier celle-ci. La question que se pose le concepteur est :
  - Comment écrire la fonction ?

CONCEPTION D'UNE FONCTION

La méthode proposée ici pour concevoir et utilisé une fonction se fait en 3 étapes :

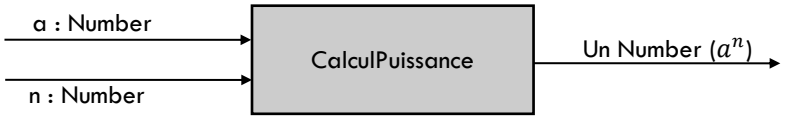
ÉTAPE 1: REPRÉSENTATION GRAPHIQUE DE LA FONCTION: IL S'AGIT DE DÉFINIR 3 CHOSES:

- Un **nom** : doit être le plus significatif possible par rapport à la tâche accomplie. C'est le cas dans l'exemple précédent avec **calculPuissance()** ou **convert_avi_to_mpeg()**
- Quels sont les **paramètres d'entrées** de la fonction ? Un paramètre d'entrée est une donnée fournie par le programme appelant à la fonction.
- Le **résultat** rendu : on précise son type. Le résultat est la valeur retournée par la fonction au programme appelant

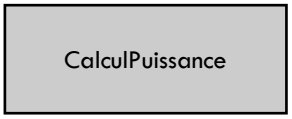


Exemple : On souhaite écrire une fonction permettant d'obtenir le carrée d'un nombre. Après réflexion nous obtenons le résultat suivant :

Dans l'exemple de CalculPuissance nous obtenons :



Remarque : on pourrait décider de se passer des paramètres (a) et (n), ainsi que de la sortie. Cela reviendrai à représenter notre fonction sous la forme d'une boîte noire sans entrées ni sortie :



Ce qui donnerait au niveau du code le résultat suivant :

```
function calculPuissance(){
   let a=0, n=0;
   a = prompt("Donnez un nombre: ");
   n = prompt("Combien de puissances successives souhaitez-vous ? ");
   let puissance = 1;
   for (let i = 1; i <= n; i++) {
       puissance = puissance * a;
   }
   document.write(a  +"<sup>" +n +"</sup>=" +  puissance);
}
```

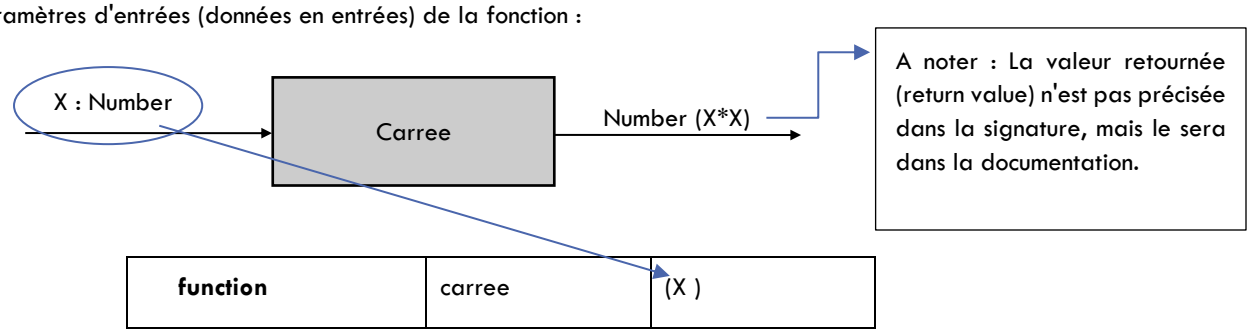
Le problème de ce choix c'est que notre fonction n'est plus **indépendante**.

L'utilisateur est obligé de faire des saisies via des boîtes de dialogue, la communication du résultat se fait par un affichage qui n'est pas personnalisable. Le jour ou la saisie du nombre et de la puissance doit se faire dans une zone texte (dans un formulaire HTML) cette fonction ne marchera plus.

A RETENIR : on évitera SAISIE et AFFICHAGE dans les fonctions, à moins que celles-ci soit dédiées à ses fonctionnalités.

ÉTAPE 2 : CONCEVOIR LA FONCTION

1. Le mot clef utilisé en JavaScript pour définir une fonction est **function**
2. La première ligne s'appelle **prototype** ou **signature.** On retrouve les informations de notre boîte noire
   - Le nom de la fonction
   - Les paramètres d'entrées (données en entrées) de la fonction :



Ce qui donne la déclaration suivante :

```
function carre(X)    {    // signature de la fonction
```

Remarque : La signature d'une fonction est très importante. C'est le mode d'emploi ou notice explicative de la fonction. Il faut donc savoir interpréter celle-ci. Il est important de prendre la bonne habitude de documenter son code ce qui reviens à donner le mode d'emploi.

La documentation de la fonction s'effectue directement dans le code à l'aide de commentaire appelait JsDoc : https://jsdoc.app/

Exemple complet avec la fonction **calculPuissance** dans un fichier "mesFonctions.js" :

```
/**
* @author Samuel Gibert
* Date de création: 18/11/2020
* Date de modification: -
```

```
* @description Cette fonction élève un nombre (a) à la puissance (n)
* @param {Number} a : le nombre a élevé à une puissance
* @param {Number} n : la puissance
* @returns {Number} : le nombre (a) élevé à la puissance (n=)
*/
function calculPuissance(a,n){  // { marque le début du corps de la fonction
    let puissance = 1;
    for (let i = 1; i <= n; i++) {
        puissance = puissance * a;
    }
    return puissance;
}  // } marque la fin de la fonction
```

3. Le corps de la fonction :

Le corps de la fonction c'est le code de la fonction proprement dit. On utilise également le vocabulaire suivant :

- Définition de la fonction
- Implémentation de la fonction

Ce code est placé entre les { } de la fonction. Voir code de la fonction **calculPuissance** en début de page.

LES PARAMÈTRES

Notre fonction :

```
function calculPuissance(a,n){     let puissance = 1;   ... }
```

Dans la signature nous voyons que la fonction a 2 paramètres : (a) et (n). Il faut considérer ces paramètres comme des variables locales, c'est-à-dire utilisable uniquement dans le corps de la fonction et pas en dehors de la fonction. La durée de vie de cette variable est la fonction elle-même. On appelle cela la portée d'une variable. Dans le cas d'une fonction, les paramètres (ici **a** et **n**) et variables définie à l'intérieur de celle-ci (ici **puissance**) ont une portée locale.

RETURN

Ce mot clef **return** permet de retourner (ou renvoyer) au programme appelant une valeur. Cette instruction quitte la fonction. Toutes instructions suivant un return ne sera jamais exécuté.

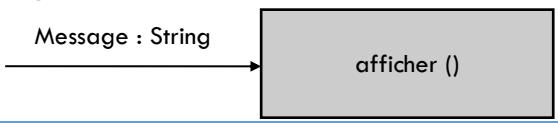
Exemple :

```
/**
 * @author Samuel Gibert
 * date de création : 18/11/2020
 * date de modification : -
 * @description La fonction carrée permet d'élevé un nombre (x) au carrée
 * @param {Number} x
 * @returns {Number} retourne x élevé au carrée x<sup>2</sup>
 */
function carree(x){
    return x*x;
}
```

Il est possible qu'une fonction ne retourne rien. C'est souvent le cas de fonctions dont le but principal est d'afficher quelque chose à l'écran. Un affichage n'est pas un résultat (**return value**) en soit. Dans ce type de fonction on terminera le programme simplement avec le mot clef **return**.

Exemple : fonction "afficher" qui écrit dans la page Web le message passé en paramètre

```
/**
 * @author Samuel Gibert
 * date de creation : 18/11/2020
 * date de modification : -
 * @description La fonction afficher permet d'écrire un message à l'écran
 * @param {String} message : le message à écrire
 * @returns {undefined} Cette fonction ne retourne rien
 */
```



```
function afficher(message){
    document.write(message);
    return;
}
```

ETAPE 3 : UTILISATION D'UNE FONCTION

Maintenant que nos fonctions (sous-programmes) sont définies dans le fichier **mesFonctions.js**, nous allons pouvoir les utiliser dans n'importe quels autres programmes. Il faut inclure pour cela ne fichier JavaScript. Il faut écrire :

```
<script src="mesFonctions.js" type="text/javascript"></script>
```

L'exécution d'un sous-programme est demandée par le programme principal : **programme appelant.** Ce programme effectue **l'appel** de la fonction, ce qui consiste à faire un saut vers la fonction. C'est comme si le programme appelant se mettait en pause, le temps de l'exécution de la fonction.

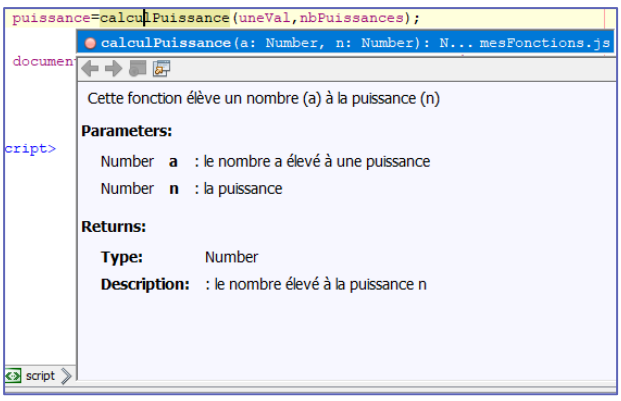
Remarque : si le sous-programme n'est jamais appelé, il ne sera jamais exécuté.

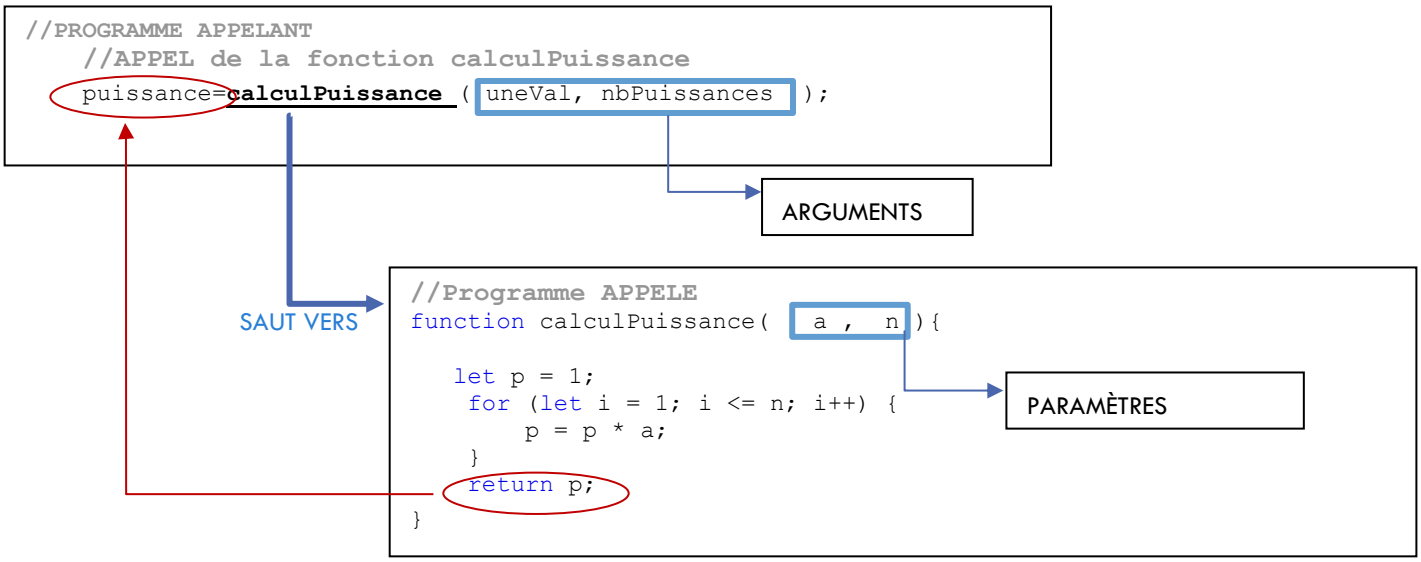
Comment faire ?

- Lors de l'appel, il doit y avoir autant d'arguments que de paramètres.
- La liste des arguments doit être du type de donnée énoncé dans la signature.
- À noter que même dans le cas de ces fonctions n'exigeant aucun argument les parenthèses () sont obligatoires après le nom de la fonction.
- Exemple de programme appelant :

```
<head>
    ...
        //Le fichier mesFonctions.js contient la fonction calculPuissance
    <script src="mesFonctions.js" type="text/javascript"></script>
</head>
<body>
    <script>

        //Le PROGRAMME APPELANT
        let uneVal = 0, puissance = 0;
        let i = 0, nbPuissances = 0;

        uneVal = prompt("Donnez un nombre: ");
        nbPuissances = prompt("Combien de puissances successives souhaitez-vous ? ");

        // Appel de la fonction calculPuissance. Les données en entrées sont uneVal et
nbPuissance
        puissance=calculPuissance(uneVal,nbPuissances);

        document.write(uneVal + "<sup>" + nbPuissances + "</sup>=" + puissance);
    </script>
</body>
```

- Comme nous avons documenté notre fonction calculPuissance, lorsque nous tampons le code de notre programme appelant "le mode d'emploi est affiché". Il nous indique les paramètres d'entrées et leur type, la valeur de retour (Returns), et donne une explication de la fonction





PARAMÈTRES ET ARGUMENTS

On note dans l'exemple précédent les termes PARAMÈTRES et ARGUMENTS. Plusieurs remarquent à ce sujet:

- Les paramètres sont des variables utilisés dans le corps de la fonction. Leur durée de vie est celle de la fonction
- Les arguments sont les valeurs ou variables utilisées dans le programme appelant.
- Les paramètres et arguments n'ont pas nécessairement le même nom.
  - Dans l'exemple (a) et (n) ont une portée locale à la fonction et utilisable dans celle-ci, mais pas dans le programme appelant.
  - Et inversement (uneVal) et (nbPuissance) sont connues dans le programme appelant, mais ne sont pas reconnues dans la fonction

Principe de fonctionnement :

Au moment de l'appel du sous-programme, les valeurs des arguments sont affectées aux paramètres

Valeur du premier argument (uneVal)	est affecté	au premier paramètre (a)
Valeur du deuxième argument(nbPuissance)	est affecté	au deuxième paramètre (n)

Les arguments sont des "Données". Le programme appelé ne peut pas modifier les arguments du programme appelant.

À la fin de la fonction, l'instruction "return p;" retourne la valeur de la variable (p) local à la fonction, et affecte celle-ci à la variable (puissance) du programme appelant.

Important :

- Le nombre d'arguments doit correspondre au nombre de paramètres
- Le type du nième argument doit correspondre au type du nième paramètre.
- L'argument doit posséder une valeur.

PORTE DES VARIABLES

Une variable est soit globale, soit locale, tout dépend de l'endroit où elle est déclarée.

VARIABLE GLOBALE

Une variable globale est une variable qui est "connue" donc utilisable à tout point du programme. Pour faire simple, en JavaScript toute variable déclaré en dehors d'une fonction est globale

VARIABLE LOCALE

Une variable locale est une variable qui est "connue" dans le bloc d'instruction ou elle est déclarée. Pour rappel un bloc d'instruction est délimité par les { }. En dehors de son bloc d'instruction, cette variable n'est pas connue. La durée de vie est celle du bloc d'instruction.

FAIRE LE BON CHOIX

Une erreur fréquente consiste à définir des variables globales pour s'éviter de réfléchir aux paramètres à utiliser dans nos fonctions. Cela fonctionne, mais c'est une solution de faciliter. En faisant ainsi nos fonctions ne sont plus indépendantes. Utiliser dans un autre contexte cela oblige à déclarer nos variables en globales. Ce n'est donc pas une bonne solution.

Conclusion : La bonne solution consiste à utiliser des paramètres.

FONCTION DÉJÀ EXISTANTE

Avant de concevoir une fonction, il faut toujours se poser la question: est-ce qu'elle n'a pas déjà été écrite ?

Si oui cela nous évitera de la réécrire.

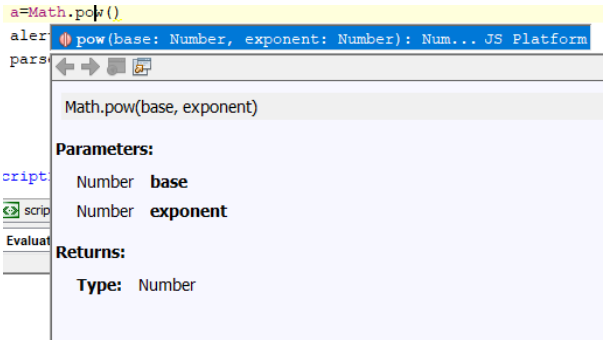
C'est par exemple le cas du calcul du sinus d'un angle. Que se passe-t-il sur une calculatrice ? On vous fournit quelques touches spéciales, dites touches de fonctions, qui vous permettent par exemple de connaître immédiatement ce résultat. Si vous voulez connaître le sinus de 35°, vous taperez 35, puis la touche SIN.

Tout langage de programmation propose de même un certain nombre de fonctions prédéfinies. C'est aussi le cas en JavaScript. Bien souvent ces fonctions se trouvent "encapsuler" dans des Objets.

Exemple : quelques fonctions définies dans l'objet Math

Fonction	Description	Exemple
sin(x)	cosinus x en radian	cos(2) renvoie –0.416147
exp(x)	exponentielle (base e)	exp(2) renvoie 7.38906
log10(x)	logarithme de x(base 10)	log10(2) renvoie 0.30103
sqrt(x)	racine carrée de x	sqrt(2) renvoie 1.141421
pow(x,p)	x à la puissance p	pow(2,3) renvoie 8.0

Dans ce tableau nous constatons qu'il existe une fonction pow. C'est l'équivalent de notre fonction calculPuissance() qui ne sert donc à rien. Voici la documentation de cette fonction :



Comme nous l'avons déjà expliqué la signature c'est le mode d'emploi de la fonction. Il est donc indispensable d'être capable d'interpréter une signature. Ci-dessous un lien pour les fonctions existantes :

https://devdocs.io/javascript/global_objects/math/pow

Voici quelques exemples de code utilisant la fonction Math.pow

```
let a=3.5;
let res;
res=Math.pow(a,2.0);  //a est un double
alert(res);   //res= a^2.0
```

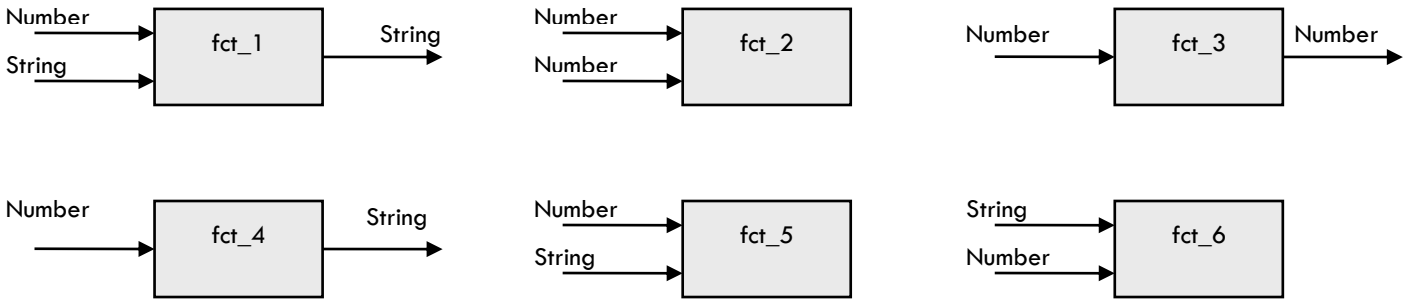
```
let res;
res=Math.pow(5.3,2);
document.write(res);
```

```
let a=3.5, b=5.2;
alert(Math.pow(a,b));
```

```
alert(Math.pow(2,Math.pow(2,2)));
```

EXERCICE

Exercice 1: donner la signature de chacune des fonctions représentées sous forme de "boites noires".



Exercice 2: pour chaque énoncé, donnez sa représentation sous forme de boite noire.

- Écrire une fonction permettant d'obtenir la valeur minimum entre 2 nombres.
- Écrire une fonction permettant de savoir si un nombre est compris dans un intervalle de 2 valeurs (min et max)
- Écrire une fonction permettant de savoir si un nombre représente un nombre binaire ou non.
- Écrire une fonction permettant de savoir si un nombre représentant une date sous la forme JJMMAAAA est valide ou non.
- Écrire une fonction permettant de calculer la remise sur un article en fonction de son prix et du taux de réduction.
- Écrire une fonction permettant d'obtenir la valeur décimale d'un nombre en Hexadécimal.
- Écrire une fonction qui affiche la date du jour selon le format suivant: "JJ-nom du mois-AAA"
- Écrire une fonction permettant d'obtenir le résultat d'une opération arithmétique simple du type "20 + 25".