COURS 3

Contexte et Evénements

BEQUART VALENTIN

Typage Dynamique
Faible

Interprété

Orienté Objet
(Prototype)

Fonctionnel et
événementiel

Inclusion dans une page Web

```
<html>
  <head>
    <!-- Les scripts sont chargés avant le body -->
    <script src="./myScript1.js"></script>
    <script src="./myScript2.js"></script>
    <script src="./myScript3.js"></script>
  </head>
  <body>
    <!-- Mon code HTML -->

    <!-- Les scripts sont chargés après le body -->
    <script src="./myScript4.js"></script>
    <script src="./myScript5.js"></script>
  </body>
</html>
```

Petits rappels

```
let monModule = (function() {

    let count = 0;

    return {
        inc: () => ++count,
        dec: () => --count,
        add: number => count += number,
        sub: number => count -= number
    }

})();
```

```
let monModule = (function () {
    function getModuleInterne(){
        console.log("fonction interne")
    }

    return {
        getModuleExterne(a){
            getModuleInterne();
            return "Fonction externe : "+a;
        }
    }
}) ();
```

Le modèle MVC

- Bonne pratique de programmation, sépare votre application en trois parties
  - Modèle (les données)
  - La vue
  - Le contrôleur

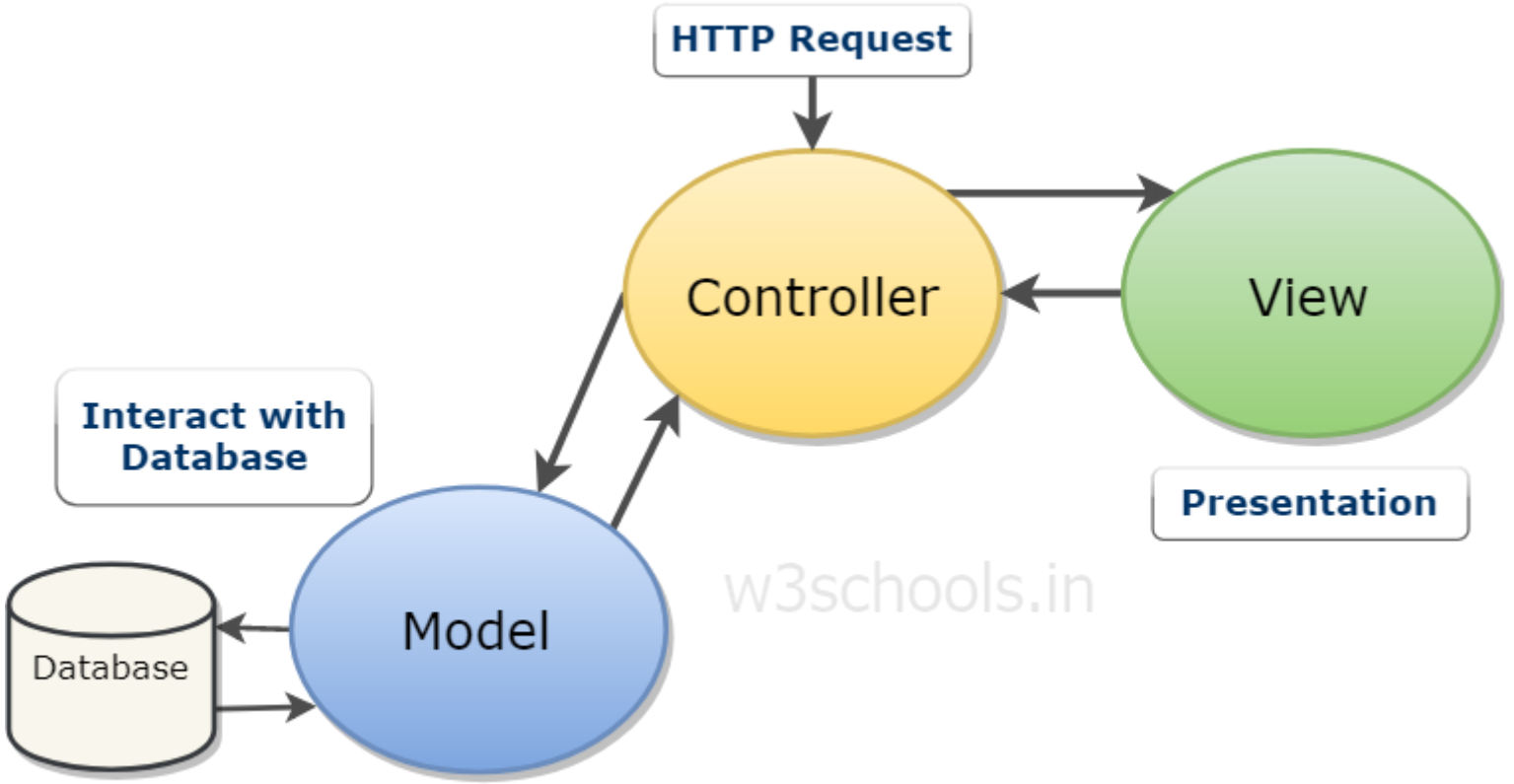
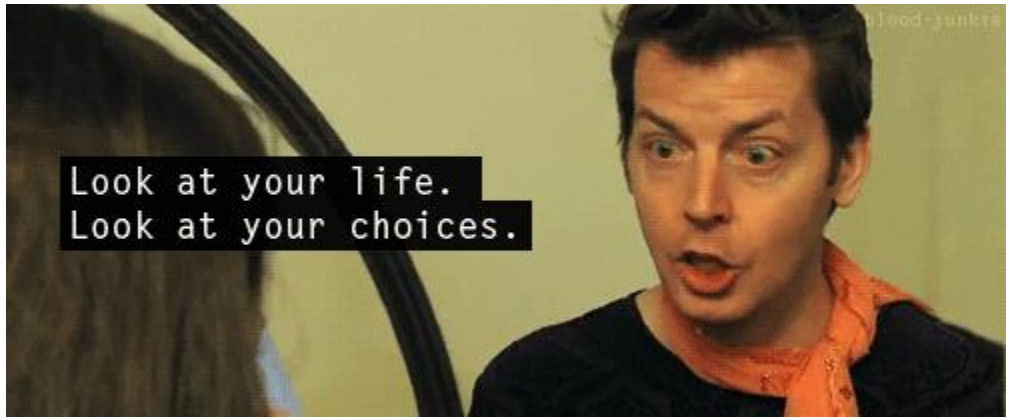


Fig: MVC Architecture

Bootstrap



Le contexte en Javascript

Une fonction est invoquée :

- A partir de son nom

- A partir d'une référence directe

- A partir d'une référence dans un objet

```
function repeat() {}
repeat();

let myFunctionReference = function {};
myFunctionReference();

let jedi = {};
jedi.myFunction = function() {};
jedi.myFunction();
```

Le contexte en Javascript

Il s'agit de l'objet à partir duquel une fonction est invoquée

Pour accéder au contexte : this

```
let jedi = {
    age:24,
    name:'Luc Skywalker'
};
jedi.display = function() {
    console.log(this.name, 'is', this.age);
};
jedi.display();
```

Le contexte en Javascript

Donc des objets différents : Contexte différents : Rappelez vous les closures!

```
let display = function() {
    console.log(this.name, 'is', this.age);
};

let bob = { age:42, name:'Bob', display: display };
let alice = { age:22, name:'Alice', display: display };

bob.display();
alice.display();
display();
```

Window

Le mot clé window permet d'accéder au contexte global

```
let globalVariable = 10;
console.log(globalVariable);                    // 10
console.log(window.globalVariable);             // 10
console.log(window.window.globalVariable);      // 10
console.log(window.window.window.globalVariable); // 10
console.log(window === this);                   // true
```

Remember

Vous vous souvenez de apply?

Call et Apply permettent d'imposer un contexte!

```
let display = function() {
    console.log(this.name, 'is', this.age);
};

let bob = { age:42, name:'Bob' };
let alice = { age:22, name:'Alice' };

display.call(bob);  // "Bob is 42"
display.call(alice);    // "Alice is 22"
```

Call

La fonction call:

Prend en paramètre le contexte et les arguments

```
let reaction = function(emotion, period) {
    console.log(this.name, 'is', emotion, 'this', period);
};

let valentin = { name:'Valentin' };

reaction.call(valentin, 'happy', 'morning');    // "Valentin is happy this morning"
reaction.call(valentin, 'hungry', 'afternoon'); // "Valentin is hungry this afternoon"
reaction.call(valentin, 'thirsty', 'evening');  // "Valentin is thirsty this evening"
```

Apply

La fonction apply:

Prend en paramètre le contexte suivi d'un tableau contenant les potentiels arguments.

```
let reaction = function(emotion, period) {
    console.log(this.name, 'is', emotion, 'this', period);
};

let valentin = { name:'Valentin' };

reaction.apply(valentin, ['happy', 'morning']);    // "Valentin is happy this morning"
reaction.apply(valentin, ['hungry', 'afternoon']); // "Valentin is hungry this afternoon"
reaction.apply(valentin, ['thirsty', 'evening']);  // "Valentin is thirsty this evening"
```

Bind

La fonction bind :

Permet d’associer un contexte à une fonction.

```
let reaction = function(emotion, period) {
    console.log(this.name + ' is ' + emotion + ' this ' + period);
}

let valentin = { name: 'Valentin' };
let gabriel = { name: 'Gabriel' };

let valReaction = reaction.bind(valentin);
valReaction('hungry', 'afternoon'); // Valentin is hungry this afternoon
valReaction('thirsty', 'evening'); // Valentin is thirsty this evening

let gabReaction = reaction.bind(gabriel, 'happy');
gabReaction('afternoon'); // Gabriel is happy this afternoon
```

Bind

Une fonction bindée ne peut plus changer de contexte!

```
let reaction = function(emotion, period) {
    console.log(this.name + ' is ' + emotion + ' this ' + period);
}

let valentin = { name: 'Valentin' };
let gabriel = { name: 'Gabriel' };

let valReaction = reaction.bind(valentin);
let gabReaction = valReaction.bind(gabriel);

gabReaction('hungry', 'afternoon'); // Valentin is hungry this afternoon
```

Problème : Les callback

Ici, la fonction forEach dispose d'un callback.

Dans un callback, le contexte est modifié

Le contexte n'est plus valentin.


```
ute', 'Le Truand'],

ction(title) {
do, title);

// Affiche :
    // undefined Le Bon
    // undefined La Brute
    // undefined Le Truand
```

Sauveur n°1

Solution: la binder au contexte dans lequel elle est sensée se trouver.

```
let valentin = {
    pseudo: 'Valentin',
    titles: ['Le Bon', 'La Brute', 'Le Truand'],
    display: function() {
      this.titles.forEach(function(title) {
        console.log(this.pseudo, title);
      }.bind(this));
    }
  };

  valentin.display();
  // Affiche :
      // Valentin Le Bon
      // Valentin La Brute
      // Valentin Le Truand
```

Sauveur n°2 : de vieilles amies

Autre solution: Les arrow function!

Elles ne modifient pas le contexte!

En effet, une arrow function utilise le contexte
dispo n
clas


```
let valentin = {
    pseudo: 'Valentin',
    titles: ['Le Bon', 'La Brute', 'Le Truand'],
    display: function() {
        this.titles.forEach(title => console.log(this.pseudo, title));
    }
};

valentin.display();
// Affiche :
    // Valentin Le Bon
    // Valentin La Brute
    // Valentin Le Truand
```

Les événements DOM

Un évènement :
Message envoyé depuis le navigateur pour informer d'un changement d'état d'un objet du DOM

Il faut donc écouter ces messages si l'on souhaite les récupérer.

Les listeneurs

```
let toto = document.getElementById('toto');
toto.onclick = myFunction;      // Un seul listener...

let toto = document.getElementById('toto');
toto.attachEvent('click', myFunction);    // Uniquement IE < 9...

let toto = document.getElementById('toto');
toto.addEventListener('click', myFunction);  // Standard W3C!!!
```

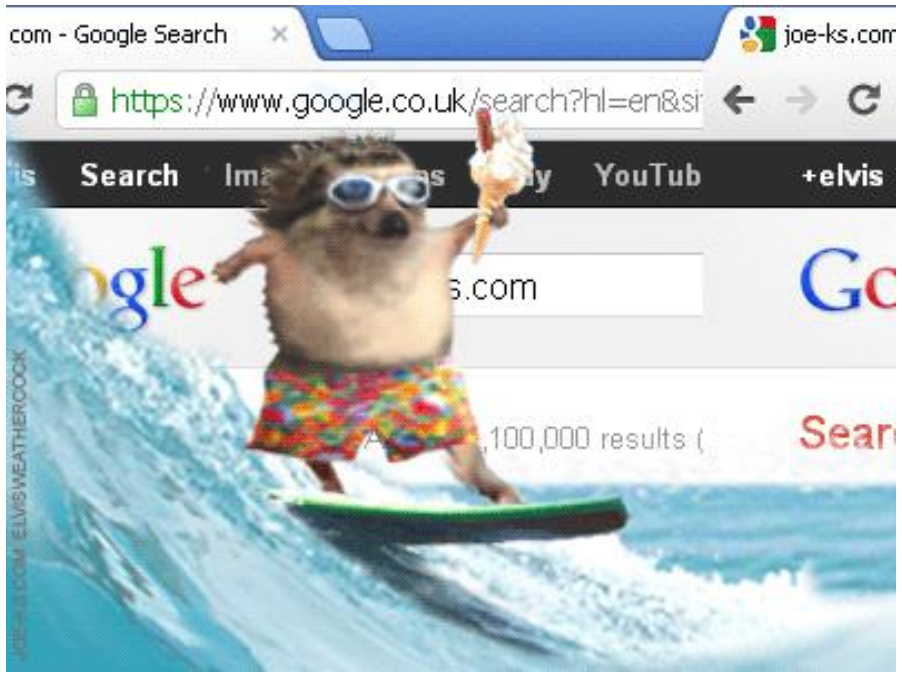
Les listeneurs (suppression)

```
let toto = document.getElementById('toto');
toto.onclick = null;      // ...

let toto = document.getElementById('toto');
toto.dettachEvent('click', myFunction);    // Uniquement IE < 9...

let toto = document.getElementById('toto');
toto.removeEventListener('click', myFunction);  // Standard W3C!!!
```

Les évènements



L’objet Event

Il possède un ensemble d’attributs et de méthodes (as usual):

- Type
- currentTarget
- Target
- timeStamp
- defaultPrevented
- cancelable
- eventPhase
- Bubbles
- preventDefault()
- stopPropagation()
- stopImmediatePropagation()

Liste des événements

- Ressources : load, unload, error, ...
- Focus : focus, blur
- Formulaires : reset, submit
- Vues : resize, scrool
- Presse-papier : cut, copy, paste
- Clavier : keydown, keyup
- Souris : click, dblclick, mouseover, select...
- Drag&Drop : drag, drop, dragend, dragstart, ...
- ...

L’objet Window

L’objet Window représente la fenêtre contenant le document (DOM) actuellement affiché.

Il permet d’obtenir plusieurs informations concernant l’état de cette fenêtre :

- Si la fenêtre a fini de charger
- Le navigateur
- L’url
- La taille de la fenêtre

Il existe plusieurs propriétés, plus ou moins utiles selon les contextes :

https://developer.mozilla.org/fr/docs/Web/API/Window

Load

```
window.addEventListener("load", event => {
    console.log("Toutes les ressources sont chargées !");
});
```

Evènement pour détecter la fin du chargement de la page

Click

```
let listOfLiElements = document.getElementsByTagName('li');

for (const liElement of listOfLiElements) {
    liElement.addEventListener('click', event => {

        event.currentTarget.classList.toggle('li-approved');
        event.currentTarget.classList.toggle('li-not-approved');

    });
}
```

Evènement pour
détecter un clic sur
un élément

Focus et Blur

Evènement pour détecter le focus sur un élément et la fin de son focus (blur donc)

```
let textFieldElement = document.getElementById('textField');

textFieldElement.addEventListener('focus', event => {
    let pTypping = document.getElementById('typing');
    let pNotTypping = document.getElementById('not-typing');
    pTypping.classList.remove('p-hidden');
    pTypping.classList.add('p-shown');
    pNotTypping.classList.remove('p-shown');
    pNotTypping.classList.add('p-hidden');
});

textFieldElement.addEventListener('blur', event => {
    let pTypping = document.getElementById('typing');
    let pNotTypping = document.getElementById('not-typing');
    pNotTypping.classList.remove('p-hidden');
    pNotTypping.classList.add('p-shown');
    pTypping.classList.remove('p-shown');
    pTypping.classList.add('p-hidden');
});
```

Change

```
textFieldElement.addEventListener('change', event => {
    console.log('L\'utilisateur a saisi :' + event.target.value);
});
```

Evènement pour détecter le changement de valeur d’un élément

Scroll

```
let scrollElement = document.getElementById('userScroll');
let lastScrolledPosition = 0;

scrollElement.addEventListener('scroll', event => {

    lastScrolledPosition = event.currentTarget.scrollTop;
    console.log(lastScrolledPosition);

});
```

Evènement pour
détecter le scroll
d'un élément

MouseEnter et MouseLeave

```
scrollElement.addEventListener('mouseleave', event => {
    let pDiv = document.getElementById('userInDiv');
    pDiv.innerText = "L'utilisateur quitte la div";
});

scrollElement.addEventListener('mouseenter', event => {
    let pDiv = document.getElementById('userInDiv');
    pDiv.innerText = "L'utilisateur est dans la div";
});
```

Evènement pour
détecter le passage
de la souris dans
un élément et son
départ

Event.preventDefault()

Permet d’arrêter le comportement par défaut d’un élément.

```
<h2>Démonstration du preventDefault</h2>

<label for="check">Checkbox:</label>
<input type="checkbox" id="check"/>
```

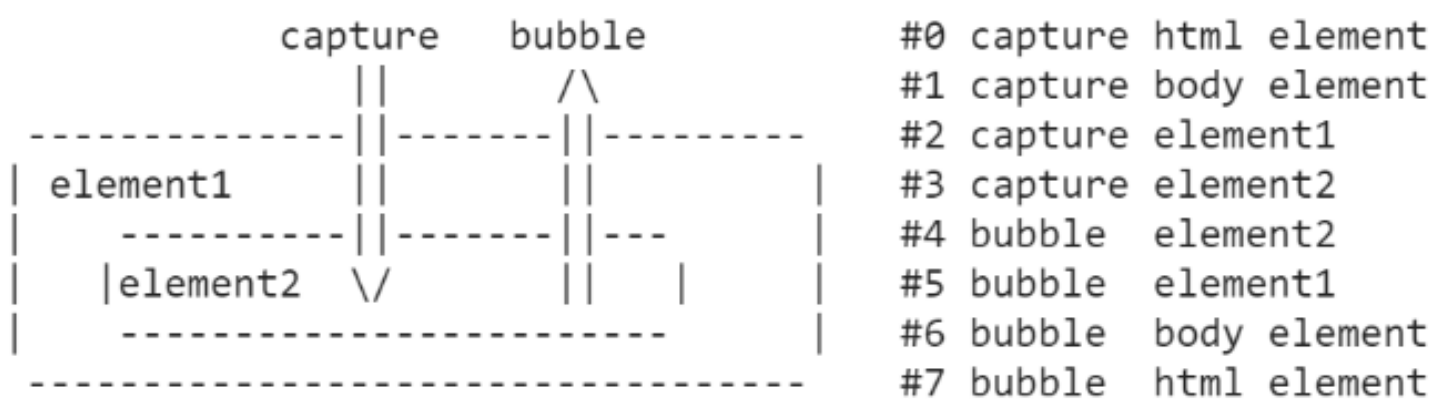
```
let check = document.getElementById('check');

check.addEventListener('click', event => {
    console.log('non non non!');
    event.preventDefault();
});
```

La propagation

Les événements se propagent suivant deux phase: Capturing et Bubbling

Un événement part du noeud élément HTML et se propage vers le noeud à l'origine de l'événement (capture) puis retourne vers l'élément racine HTML (bubble).



Exemple de propagation

```html
<h2>Démonstration de la propagation</h2>
<div id="propaDiv">
    <form id="propaForm">
        <input id="propaButton" type="button" value="Click">
    </form>
</div>
```

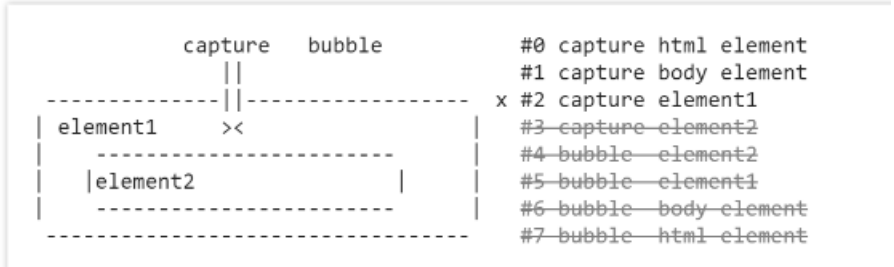
```javascript
let propaDiv = document.getElementById('propaDiv');
let propaForm = document.getElementById('propaForm');
let propaButton = document.getElementById('propaButton');

propaDiv.addEventListener('click', event => {
    alert('Click sur Div');
});

propaForm.addEventListener('click', event => {
    alert('Click sur Form');
});

propaButton.addEventListener('click', event => {
    alert('Click sur Button');
});
```

Stopper la propagation



Il existe des cas où l'on ne souhaite pas que l'élément se propage.

Dans ce cas, on peut utiliser la fonction event.stopPropagation()

```
element1.addEventListener('type', function(event) {
    event.stopPropagation();
}, true);
```

Exemple de propagation stoppée

```html
<div id="propaDivScd">
    <form id="propaFormScd">
        <input id="propaButtonScd" type="button" value="Click">
    </form>
</div>
```

```javascript
let propaDivScd = document.getElementById('propaDivScd');
let propaFormScd = document.getElementById('propaFormScd');
let propaButtonScd = document.getElementById('propaButtonScd');

propaDivScd.addEventListener('click', event => {
    alert('Click sur Div');
});

propaFormScd.addEventListener('click', event => {
    alert('Click sur Form');
    event.stopPropagation();
});

propaButtonScd.addEventListener('click', event => {
    alert('Click sur Button');
});
```

C’est l’heure de reprendre le TP!

Exercice (par Binôme)

- Nous allons réaliser un site présentant une calculatrice.
- On se laisse un concept simple, doivent être présent :
  - Les boutons pour les chiffres
  - Les boutons d'opérations de base : +, -, *, /, =
  - Vérification de la cohérence des nombres entrés
  - Une touche « . » pour la virgule

  Bonus :
  - Un bouton pour tracer la courbe de ces fonctions une fois terminée
  - Un bouton reset pour revenir à l'état initial
  - Une une touche « : » pour la saisie des fonctions

Saisie utilisateur

- Pour les nombres
  - Soit au clic sur un bouton
  - Soit l'entrée sur le clavier, en ayant focus l'input d'entrée pour les résultats
    - Les deux doivent être gérés
- Pour les opérations
  - Uniquement via le clic sur bouton

Les opérations de base

- Uniquement lorsque la calculatrice est en état initial (Prévoyez un indicateur pour le dire si vous vous lancez dans les graphes)
  - L'utilisateur tape ses nombres (entier ou float)
  - A l'appui sur un opérateur de base, le nombre est stocké dans une variable, ainsi que l'opération à effectuer
  - L'utilisateur tape l'autre nombre
    - Soit il fait égal, le résultat s'affiche
    - Soit il appuie sur un autre opérateur, le calcul se fait, et l'opération se fera entre le résultat et le nombre qui va être tapé (restons simple)

Techniquement

- Bien découper son code! Une nouvelle fois, un fichier main.js en point d'entrée puis, ma préconisation :
  - Un Module qui instancie les listeneurs (saisies et opérations)
  - Un Module qui prépare l'affichage des résultats à l'écran
  - Un Module qui sauvegarde les nombres entrés, la fonction demandée, l'état de la calculatrice
  - Un Module qui contient toutes les opérations que la calculatrice connait

  Bon courage!!

- Bonus : Pour le traçage des graphes, la bibliothèque Chart.js, ils ont de très bons tutos pour être inclus facilement dans un projet! Vous allez découvrir les librairies.

Les fonctions (Bonus)

- A l'appui sur l'une des touches fonctions, la calculatrice passe en mode « fonction »
  - Chaque nombre sera stocké dans un tableau d'entrée
  - Une fois dans ce mode, l'utilisateur saisi ses nombres, et appui sur la touche « = » pour valider et saisir le nombre suivant
  - Si l'utilisateur tape un nombre, appuie sur « : », et tape un autre nombre, tous les nombres (avec un pas de 1) compris dans cet intervalle sont stockés dans le tableau
  - Pour exécuter la fonction sur le tableau d'entrée et afficher la courbe, appuyer sur la touche de traçage.
  - Une fois la fonction tracée, la calculatrice revient à son état initial.