# COURS

Controle continu integral -> 50% rendu TD
50% projet

## Client - serveur

Application web <= site web dynamique

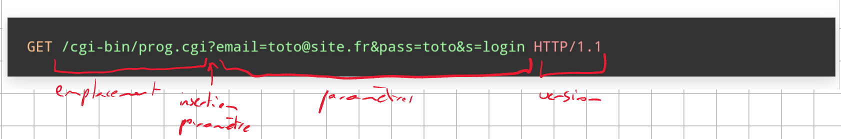
Revoir cours maison

mais vraiment !!! faire [illegible] [illegible] avec le pdf (voir Moodle)

## Le protocole HTTP

HyperText Transfer Protocol
fonctionne en mode client - serveur
protocole sans état

methode GET :



```
GET /cgi-bin/prog.cgi?email=toto@site.fr&pass=toto&s=login HTTP/1.1
```

Fin CM 1 -> composants

# Coté client (JS / Typescript !?)

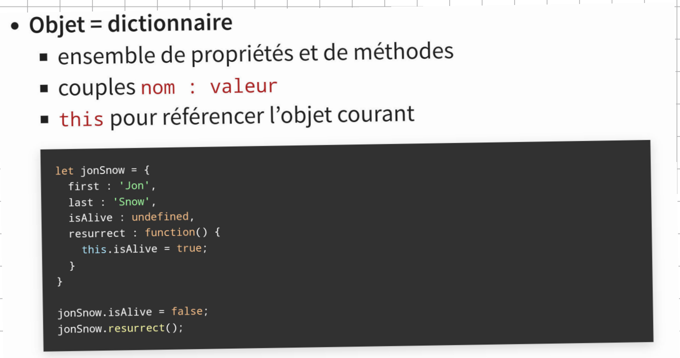
navigateur web
- moteur de rendu : HTML -> DOM -> Affichage

## Moteurs JS

- Interpréter
- Nouvelles versions (ES 7-11, Typescript)

## Bases de JS

- typage dynamique
- flexible et permissif
- langage fonctionnel
- langage OOP (object-orienté prototype)
- langage évènementiel

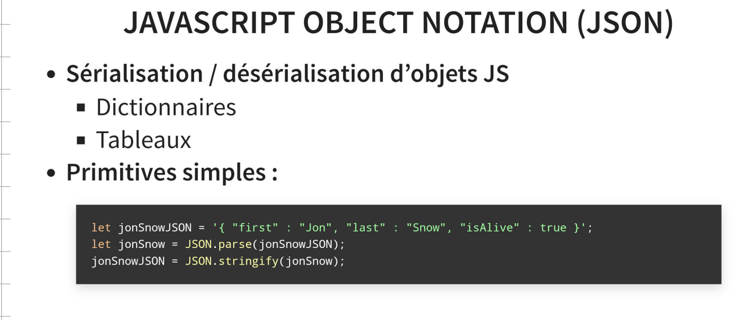


- **Objet = dictionnaire**
  - ensemble de propriétés et de méthodes
  - couples `nom : valeur`
  - `this` pour référencer l'objet courant

```
let jonSnow = {
  first : 'Jon',
  last : 'Snow',
  isAlive : undefined,
  resurrect : function() {
    this.isAlive = true;
  }
}

jonSnow.isAlive = false;
jonSnow.resurrect();
```

- Programmer de l'**orienté object** (ES6)
  - `class`, `constructor`, `static`, `extends`, `super`
  - Propriétés publiques
  - Pas de classes abstraites ou d'interfaces

```
class Person {
  constructor(first, last) {
    this.first = first;
    this.last = last;
    this.isAlive = undefined;
  }

  resurrect() {
    this.isAlive = true;
  }
}

let jonSnow = new Person('Jon','Snow');
```



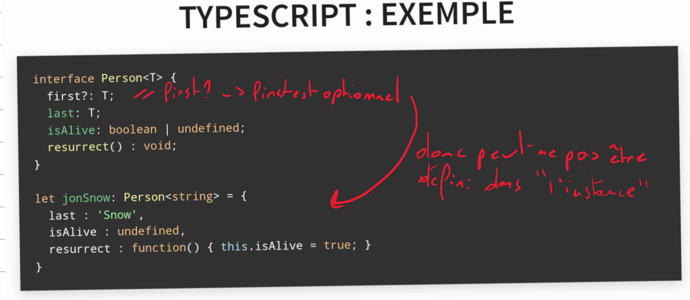
# JAVASCRIPT OBJECT NOTATION (JSON)

- **Sérialisation / désérialisation d'objets JS**
  - Dictionnaires
  - Tableaux
- **Primitives simples :**

```
let jonSnowJSON = '{ "first" : "Jon", "last" : "Snow", "isAlive" : true }';
let jonSnow = JSON.parse(jonSnowJSON);
jonSnowJSON = JSON.stringify(jonSnow);
```

## Typescript

- typage explicite
- inférence de type
- POO
- généricité
- modularité



# TYPESCRIPT : EXEMPLE

```
interface Person<T> {
  first?: T;
  last: T;
  isAlive: boolean | undefined;
  resurrect() : void;
}

let jonSnow: Person<string> = {
  last : 'Snow',
  isAlive : undefined,
  resurrect : function() { this.isAlive = true; }
}
```

// first? -> first est optionnel

donc peut ne pas être défini dans "l'instance"

Pleins d'exemples après...

Programmation réactive :

watchEffect() ≡ computed()

↓ met à jour dynamiquement une valeur

↓ résultat mise en cache

Programmation événementielle

- Objet DOM Event
  - click, load, mouseover, submit, ...
- Deux processus en parallèle
- Deux syntaxes (attributs HTML : onclick, ...) et (eventListeners en JS)

Fonctions de rappel (callback)

- fonction passée en paramètre à une autre fonction

Promesses (Promise)

- asynchrone ou différé
- appel promesse :
  - .then() si réussi
  - .catch() si erreur
  - .finally() exécuté dans tous les cas

  } possibilité de chaîner les appels
- async / await
  - async : faire retourner une promesse
  - await : faire attendre le résultat d'une promesse (dans une fonction async)

↓

éviter d'utiliser

Requête asynchrone au serveur

- deux solutions : AJAX et Fetch API

UwU

UwU