Cours 1. *URI et page Web*

Habituellement un navigateur présente au moins deux zones : une zone en rectangle destinée à afficher chaque page web (page web : une fenêtre graphique avec texte, image, bouton), et une zone appelée barre d'adresse destinée à accueillir un texte au format URI (Uniform Resource Identifier : URI format qui comprend aussi les adresses URL) qui identifie une "ressource".

Lorsque l'utilisateur entre et valide (avec la touche entrée ou autre) une URI dans la barre d'adresse, alors le navigateur tente de récupérer la ressource désignée par cette URI. Le cas échéant alors la ressource est une liste d'octets, si elle représente une page web alors on dit que c'est un document Web le navigateur l'interprète pour construire et afficher la page.

Exercice 2. *Première page*

Afficher la page web située à l'adresse URL suivante `https://www.wikipedia.org/`.

Cours 3. *Ressource : liste d'octets*

Un document Web n'étant qu'une liste d'octet il peut par exemple être placé dans un fichier.

Exercice 4. *file :///*

Placez la ressource située à l'URL `https://defelice.up8.site/devwebbdd.html` dans un fichier maPage (exemple de commande `$wget https://defelice.up8.site/devwebbdd.html -O maPage`). Ensuite tentez de l'interpréter avec votre navigateur en indiquant `file:///chemin/vers/emplacement/maPage` comme dans la barre d'adresse (avec firefox vous pouvez aussi directement l'ouvrir avec `$ firefox chemin/vers/mapage`).

Cours 5. *Html*

Un document web représente un texte, encodé dans un format qui étend l'ASCII (ut8,latin1,...). Le texte peut être écrit selon le format html (html5 ou html version 5 en 2026). Le format html5 est décrit à cette adresse : `https://html.spec.whatwg.org/multipage/`.

Exercice 6. *code source*

Afficher la page web de `https://linux.developpez.com/formation_debian/formation-linux.html` récupérer son code source à l'aide du menu du navigateur (ou Ctrl+U dans firefox). Comparez-le avec le fichier obtenu (avec wget par exemple) et un editeur. Est-ce le même texte ? Est-ce ce que le format d'encodage de la ressource récupérée est la même que celle du texte fourni par le navigateur ? Essayer de copier le texte fournit par le navigateur et de l'interpréter avec le navigateur.

Cours 7. *Html minimum*

Le contenu minimal du texte d'un document html est le suivant :

```
<!DOCTYPE html>
<html lang="fr">
   <head>
      <title>Titre page</title>
   </head>
</html>
```

Depuis quelques temps, la norme indique que l'encodage du texte doit être `utf-8`. `fr` est à modifier selon la langue du document (influence entre autre certaines règles de mises en page). `Titre page` remplace le titre la page (qui entres autres apparait dans le titre de la fenêtre du navigateur). Celui-ci doit contenir au moins un caractère non blanc.

Exercice 8. *Exercice*

Écrire et afficher avec votre navigateur une page dont le titre est `Ma première page`.

Cours 9. *validateur*

Pour vérifier si un document html respecte la norme html5 vous pouvez utiliser `https://validator.w3.org/`.

Exercice 10. *Exercice*

Quelle(s) sont l(es) erreur(s) de syntaxe html ce document (non Web donc) ?

```
<!dOcTyPe HtMl>
<hTMl LanG="Fra">
   <HeAD>
      <tItle>Ma Page É</TitLe>
   </ heAd >
</HTML>
```

Cours 11. *Encodage*

Si vous souhaitez utiliser un autre encodage, par exemple `latin1` vous devez indiquer celui-ci avec une ligne comme `<meta charset="latin1">` placé entre `<head>` et `</head>`.

Cours 12. *Balise*

Selon la norme, un document html est structuré par des balises. Par exemple `<html lang="fr">` et `</title>` sont des balises. Chaque balise possède un type choisi parmi un nombre limité, voici quelques types `title`, `html`, `head`, `body`. Chaque type est soit solitaire soit double. Les balises de type double (comme le type `html`) vont par deux, une balise ouvrante (par exemple `<html lang="fr">` et une balise fermante qui commence par `</` (comme `</html>`). Les balises de type solitaire sont seules (comme `<br>`) on peut aussi la terminer avec `/>` (comme `<br />`). Idéalement, on ne peut pas fermer une balise avant d'avoir associé une balise fermante à tout balise ouvrante écrite depuis. On ne peut non plus écrire une balise fermante qui n'est pas associé à une balise non ouverte.

Exercice 13. *Enchevetrement*

Quel est le problème avec ce document ?

```
<!doctype html>
<html lang="fr">
   <head>
      <title>ma page</title>
   </head>
   <body>
      <p>Il ne faut pas fermer un élément après avoir fermé tous les autres
         depuis. Ce serait une <strong>erreur de norme.</p></strong>
   </body>
</html>
```

Cours 14. *body*

Les informations qui apparaitront dans la fenêtre graphique sont situé entre le couple de balises `body` qui n'est présent qu'une fois et doit être situé après `head` On y trouve par exemple la balise `p` qui représente un paragraphe qui peut être utilisée placée plusieurs fois.

Cours 15. *commentaire*

Un commentaire html débute par `<!--` et se termine au premier `-->` rencontré. Il n'existe pas de commentaire de fin de ligne ni la possibilité d'imbriquer les commentaires.

Cours 16. *Attributs*

Certaine balise ouvrante ou seule peuvent avoir une liste d'attribut qui est un couple `nom d'attribut` quelquefois associé à une valeur située indifféremment entre `"` ou `'`, par exemple `lang='fr'`. Les noms d'attributs possibles par type de balise son en nombre limité. Une balise fermante ne possède pas d'attribut.

Cours 17. *Échappement*

Le texte présent hors balise et comme valeur d'attribut ne peut utiliser certains caractères comme `<` ou `>` ou `"` ou `'`. On peut utiliser la syntaxe `&lt;` remplacé par `<`. `&gt;` remplacé par `>`, `&apos;` remplacé par `'` `&quot;` remplacé par `"`. Il existe d'autres codes. On peut aussi indiquer n'importe quel caractère de la norme unicode avec `&#d;` ou d est le code unicode du caractère écrit en décimal. où `&#xh;` où h est le code unicode écrit en hexadécimal.

Exercice 18. *sans*

Écrire une page qui n'utilise que des caractère ASCII, dont le titre est `Ma première page`. Et qui contient affiche le texte `<p> ceci est un paragraphe </p>`.

Cours 19. *Entorse*

Certaines entorses aux règles de syntaxe sont permises par la norme nous ne les décriront pas toutes. Par exemple les noms de type des balises (comme `html`, `br`) et les noms d'attributs et `<!doctype html>` ne sont pas sensibles à la casse. De plus certaines balises qui normalement vont par deux peuvent ne posséder que l'ouvrante (l'autre est implicite).

Cours 20. *type Balise*

- `p` (double) pour encadrer un paragraphe
- `ul` pour encadrer une liste non ordonnée
- `ol` pour encadrer une liste ordonnée
- `h1` pour encadrer un titre, h2 titre de seconde catégorie h3,h4,...
- `pre` (double) pour indiquer une partie brute
- `table` (double) pour ajouter un tableau
- `img` (simple) pour inclure une image attribut `src`
- `strong` (double) pour mettre un texte en relief car important
- `em` (double) pour mettre un texte en relief (autre raison que important).
- `del` (double) pour indiquer une mise à jour d'un texte, encadre une partie supprimée.
- `ins` (double) pour indiquer une mise à jour d'un texte, encadre une partie ajoutée.
- `hr` (simple) pour indiquer une séparation
- `br`(simple) pour indiquer une fin de ligne
- `main section footer nav` pour encadrer les différentes grande partie de la page.

Cours 21. *JavaScript*

ECMAScript est un langage de programmation exploitable dans un document web, il est plus connu sous le nom JavaScript. On peut l'écrire entre deux balises `script` à n'importe quel endroit d'un document html.

```
<script>
let a=5;
let c=a+7;
a="chaine de caractère";
alert("ceci est un message");
</script>
```

Exercice 22. *Premier programme*

Inclure le code précédent dans un document html et essayer de l'interpréter. Quel est l'effet de `alert` ?

Cours 23. *Norme*

Normalement un navigateur web doit suivre les recommandations du standard ECMAScript, disponible à cette adresse :

`https://ecma-international.org/publications-and-standards/standards/ecma-262/`. De plus, à cette norme le navigateur, doit ajouter des interfaces qui permette à un code `ECMAScript` d'agir avec un environnement on les appelle des `Web API` (dont fait partie `alert`).

Une liste non officielles de ces API est présente à :

`https://developer.mozilla.org/fr/docs/Web/API` Un autre exemple d'interaction est l'affichage de message dans la console du navigateur par exemple avec `console.log("message")`. Pour ouvrir la console F12 onglet console avec firefox, (cmd+maj+j avec safari ?, ctr+maj+j windows ?), elle est souvent accompagné d'un interpréteur.

```
// commentaire uniligne
/* commentaire
   multiligne */
let a=[1,3,5,"machin"];
a.push("boubidi"+"babidi");
let i=0;
```

```
while(i<a.length){
   console.log(a[i])
}
if(typeof(5)=="number"){
   console.log("5 est un \"number\"")
}else{
   console.log("5 n'est pas \"number\"")
}
```

Exercice 24. *Compter*

Écrire une page web qui se contente d'afficher des nombres de 0 jusqu'à 1000 dans la console.

Cours 25. *Source*

Les différents codes `script` d'une même page partagent le même environnement. On peut définir une fonction et l'utiliser ensuite. On peut également inclure du code ECMAScript à partir d'une URI avec l'attribut `src` mais dans ce cas le contenu de la balise est ignoré.

```
function maFonction(parametre1,parametre2){
   return parametre1+parametre2
}
let c=maFonction(5,2**7) // 2 puissance 7
```

Exercice 26. *Import*

Écrivez dans un fichier `monJs.js` la définition d'une fonction qui affiche bonjour dans la console, importez-la dans une page web et utilisez-la.

Cours 27. *Exécution*

Le code javascript est exécuté lorsqu'il est rencontré (avant que la page soit complètement construite). On peut inclure du code javascript comme valeur `onclick` d'un `button`. Ce code sera exécuté lorsqu'on cliquera sur le bouton (et donc peut intervertir après le chargement complet de la page).

```
<button onclick="let a=4;">Button<\button>
```

Exercice 28. *Compte*

Construire une page avec un bouton "clique moi" qui affiche dans la console le nombre de fois ou le bouton a été cliqué.

Cours 29. *Arbre Html*

Par sa syntaxe organisée par des balise, un document html peut être assimilé à un arbre dont la racine est `html`. Un nœud de l'arbre peut être soit un commentaire (entre `<!--` et `-->`,valeur : une liste de caractère) soit un élément (par un couple de balise `html`, `head`, `p`, ..., une chaîne valant : une des balise ayant des éventuels enfants), soit un texte (une liste de caractères).

Les seuls nœuds internes sont les éléments (`hmtl`,...).

Exercice 30. *Dessinez arbre v2*

Dessinez l'arbre enraciné à `ol`.

```
<ol type="a">
   <li> En premier </li>
   <li> En second </li>
   <!-- commentaire-->
   <li> En dernier </li>
</ol>
```

Cours 31. *Initialement*

L'arbre construit initialement ne correspond pas exactement

Cours 32. *Navigation*

Le type de valeur `node`, disponible en javascript, permet de représenter le document sous la forme d'un arbre. La racine (`html`) est accessible avec `document.documentElement`. Pour obtenir la liste des enfants d'un `node` on peut utiliser `.childNodes` sur un type `node` (exemple `noeud.childNodes`).

Pour obtenir le type d'un noeud on peut utiliser `noeud.nodeType` les valeurs possibles sont des entiers, dont les valeurs sont `Node.ELEMENT_NODE` `Node.COMMENT_NODE` `Node.TEXT_NODE`. Dans le cas d'un texte ou d'un commentaire on peut récupérer sa valeur avec `.nodeValue` dans le cas d'un élément son nom avec `.nodeName`.

Exercice 33. *Profondeur*

Fabriquer une page web avec un bouton qui, cliqué, affiche un parcours en profondeur de l'arborescence à partir du noeud `html`. Marquez les fin de parcours des enfants. Il est important que le parcours se fasse après la construction de la page.

Exemple :

```
element : html
commentaire : <<mon commentaire>>
element : head
texte : <<
    >>
element : title
texte :<< ma fenetre>>
finEnfants : title
finEnfants : head
element : body
...
...
finEnfants : html
```

Cours 34. *cours*

On peut aussi récupérer directement un noeud par la valeur de son attribut `id` avec `document.getElementById("monId")` On peut aussi directement modifier `.nodeValue`.

Exercice 35. *Affichage*

Construire une page avec un bouton "clique moi" qui, cette fois-ci, affiche le nombre de fois ou le bouton a été cliqué sur la fenêtre graphique.

Cours 36. *cours*

On peut créer un nouveau noeud texte avec `document.createTextNode("montexte")` et l'insérer à un endroit avec `noeudParent.insertBefore(noeudAInserer,enfantDeParent)` (`enfantDeParent pouvant valoir null` pour placer le noeud à la fin). On peut aussi créer un noeud élément par exemple avec `document.createElement("ul")`.

Exercice 37. *Enum*

Écrire une page web qui affiche la liste des entiers de 0 à 1000.

Cours 38. *Champs*

Une balise (seule) `input` ayant `"text"` pour valeur de `type` permet à l'utilisateur d'entrer du texte (par exemple `<input type="text" />`) Le texte du noeud est récupérable ou modifiable à travers sa propriété `.value` (exemple `noeud.value="par defaut"`).

Exercice 39. *Affiche champs*

Fabriquer une page web qui possède deux bouton et un champs de texte et qui affiche le contenu du texte dans la console lorsqu'on appuie sur un des deux boutons et remplit le champs avec le texte `valeur par défaut` lorsqu'on appuie sur l'autre bouton.

Exercice 40. *TP 2 : Enumere*

Écrire une page web `enum.html` qui affiche dans les cases d'une table (avec ligne et colonne) de largeur 7 colonnes, la liste des 1000 premiers carrés parfaits.

Cours 41. *Unicode*

On peut utiliser `String.fromCharCode` pour renvoyer une chaîne de caractères contenant un caractère dont le code unicode est l'argument de la fonction. Par exemple `String.fromCharCode(65)` renvoie la chaine de caractère `"a"`.

Exercice 42. *TP 2 : Table unicode*

Écrire une page `unicode.html` qui affiche une table unicode : Pour les valeurs unicode de 1 à 1000 la table contient le nombre et le caractère qui lui correspond dans la table.

Exercice 43. *TP 2 : Alerte !*

Écrire une page web `heure.html` Qui ne contient qu'un seul bouton intitulé «Cliquer ici» qui ouvre une fenêtre d'alerte affichant l'heure actuelle selon le système interprétant la page (vous pouvez consulter la liste des API Web pour obtenir l'heure).

Exercice 44. *TP 2 : Générateur*

Écrire une page web `ajoute.html` qui duplique le texte d'un paragraphe à chaque appuie sur un bouton. Le contenu du paragraphe au départ est «Texte abyssal ».

Exercice 45. *TP 2 : Articles*

Écrire une page web `articles.html` munit d'une boite dans lequel on peut entrer du texte (voir paragraphe affiche champs), d'un bouton et d'un seul paragraphe. A chaque appui sur le bouton le texte de la boite doit être ajouté en début du paragraphe et donc le fait grandir.

Exercice 46. *@Pascal*

Écrire une page web `pascal.html` qui affiche le contenu du triangle de pascal lorsqu'on clique sur un bouton. Les dimensions du triangle sont récupérées dans un formulaire présent sur la page. Par défaut (texte qui n'est pas un nombre en base dix) le triangle possède une dimension de 4.

Exercice 47. *@Deux détentes*

Écrire une page web `deuxDetente.html` avec un formulaire qui demande un nombre n, compris entre 1 et 13, puis après clic sur un bouton, affiche un second formulaire de case à cocher entre 0 et n-1. Après un second appui sur un autre bouton la page doit afficher les 300 premiers nombres qui sont congrus* aux valeurs cochées modulo n.

Remarque :

— * **a** est congru à **b** modulo **n** si le reste d'une division par **n** de **a** est égale à **b**.

— Par défaut n vaut 13.

Cours 48. *Suppression Noeud*

Pour supprimer un noeud on peut utiliser `noeudParent. removeChild (noeudEnfant)`.

Exercice 49. *Alternatif*

Faire une page web avec un bouton qui lorsqu'on clique sur le bouton fait disparaître ou apparaître le texte Robert-Houdin.

Cours 50. *Valeur attribut*

Pour changer la valeur d'un élément on peut utiliser `.setAttribute` par exemple `noeud.setAttribute("src","./monFichier.png")`.

Exercice 51. *Type de liste*

Écrire une page web avec un bouton et une liste, en cliquant sur un bouton, la numérotation de la liste change (`type="a"`, `type="i" type="A"`).

Cours 52. *contenteditable*

Certains attributs sont dit héréditaires : la valeur qu'on leur donne dans une balise (nom balise : `html`, `body`, `p`) est transmise aux enfants de cet élément. C'est le cas de `contenteditable` prenant la valeur `true` ou `false`,

qui permet à l'utilisateur de modifier les noeuds textes enfants. D'autres attributs ne le sont pas, comme `src` ou `id`.

Cours 53. *Pas d'hérédité avec javascript*
La transmission héréditaire de la valeur d'un attribut héréditaire. N'a pas lieu si cet attribut est modifié avec javascript.

Exercice 54. *Vérifiez-le simplement grâce à la console*

Cours 55. *Attribut sans valeur*
Certains attributs n'ont pas besoin de valeur comme `hidden`, en écrire dans une balise a pour effet de ne pas prendre en compte la balise et ses enfants dans l'affichage.

Cours 56. *Autres opérations*
A partir d'un noeud `noeud .getAttributeNames()` renvoie la liste des attributs actuels de la balise, `noeud .getAttribute (nomAttribut)` récupére la valeur actuelle de l'attribut (de type chaine de caractère), `noeud. removeAttribute (nomAttribut)` supprime un attribut actuellement présent.

Cours 57. *Styles*
Certains attributs spéciaux ont pour effet de modfier la manière dont le contenu est affiché, leur syntaxe est particulière . On les appelles les attributs de style.

Cours 58. *Border*
Pour afficher un cadre autour du contenu d'un élément affiché on utilise l'attribut de style `border`. Pour changer la couleur du texte on utilise l'attribut de style `color`. Par exemple on peut écrire `style="border: solid 1px red; color:blue"`

Exercice 59. *px*
Que signifie `px` ?

Cours 60. *<style>*
On peut appliquer un style à plusieurs balises à la fois en plaçant des instructions ayant cette forme `(selecteur) { (style appliqué) }` entre des balises `<style>` dans `<head>`. Par exemple le texte `p color:blue; background-color: red;` va donner à toute les balises `p` le style contenu entre les accolades.

Exercice 61. *Petites boîtes très étroites*
Le système de mise en page peut compris à l'aide de rectangles appelé boites. En utilisant le selecteur universel `*` (qui selectionne tout élément) mettez en évidence la disposition des boîtes d'une page.

Cours 62. *style javascript*
Pour modifier un attribut de style d'un noeud on peut utiliser `.style` par exemple `noeud.style["color"]="blue;"`.

Exercice 63. *Clignotant*
Construire une page avec un bouton qui fait changer la couleur d'un texte à chaque clic sur celui-ci.

Cours 64. *Cours (bilan style)*

1. `display:none`, pas de rendu pour cet élément.
2. `width:70px` force le contenu de la boite à 70 pixels.
3. `height:2cm` force le contenu interne de la boite à 2cm.
4. Si le contenu dépasse on peut indiquer comment le traiter avec `overflow`, qui peut valoir `hidden scroll`.
5. `position:fixed` La boite n'a plus de physique, elle est placée à un emplacement fixe par rapport haut gauche de la fenêtre graphique. Pour la position on peut utiliser par exemple `top:-5px` et `left:30cm` (top :0px et left :0px n'est pas forcément tout en haut tout à gauche).
6. `position:absolute` La boite n'a plus de physique, elle est placée (avec `top` et `left` par exemple) à une position fixe par rapport à la page web.

7. `position:relative` La boite possède une présence physique mais son rendu peut être décalé par rapport à cette position, si `top:0px` et `left:0px` ou rien alors pas de décalage.
8. `visibility:hidden` pas de rendu (mais ne change pas l'occupation physique).
9. `display:contents` indique que l'élément est ignoré en tant que boite et les enfants ajoutés directement comme enfants du parent de l'élément (par exemple on peut cependant indiquer une couleur au texte mais pas au background).
10. `display:inline flow` similaire à `contents` mais l'élément n'est pas complétement ignoré on peut donner un background par exemple (cependant certaines actions comme lui donner une taille sont sans effet).
11. `display:inline flow-root` le contenu produira une boite qui s'ajoutera comme un mot dans le flux de texte. On peut changer la manière dont les éléments s'agence à l'intérieur de la boite en changeant flow-root.
12. `display:inline flex` chaque enfant est t enfilé dans une boite.
13. `display:inline grid` Les enfants sont alignés sur une grille.
14. dans les cas précédents on peut choisir entre `inline` et `block` pour indiquer comment la boite elle-meme se comporte dans un flow. Si c'est block elle se comporte comme une ligne, inline elle se comporte comme un mot.
15. Mise en page pour `flow` `text-indent:<dist>` : indentation de la première ligne du paragraphe. `text-align` : justification : `right, center, left, end, start, justify` `word-spacing:<dist>` espace entre mots (par rapport à l'espace de la police). `letter-spacing:<dist>` espace entre lettres d'un mot. `line-height:<dist>` espace entre lignes (valeur positive absolue).
16. Décoration d'une boite : aux dimensions interne s'ajoute dans cet ordre (de l'interieur vers l'exterieur) `padding:` : espace entre le cadre et contenu `border` : espace coloriable exemple `solid <dist> <color>` `margin:` : espace entre le cadre et le bord physique exterieur `outline:` : espace coloriable `background-color` : couleur du cadre

    Pour chacun on peut préciser le bord, par exemple avec `padding-top`, `margin-left ...-bottom`. `border-radius <dist>` pour arrondir un cadre.
17. La police : `font-size:<dist>` `color:<col>` `text-decoration: overline <color> underline dotted` `text-transform: uppercase lowercase capitalise` `font-weight: bold` `font-style: italic` `font-family`, pour ajouter une police `@font-face{font-family: <identifiant> ; src:url(".....")`
18. Le traitement des caractères blancs et sauts de ligne `white-space:` valeurs `normal pre nowrap pre-wrap preline`
19. `transform: matrix3d(<4x4>) rotateX(...) rotateZ(...)`

Cours 65. *Selecteur javascript*

On peut voir un selecteur comme une fonction qu'on applique à un arbre et qui renvoie un ensemble de noeuds. Il existe en javascript la fonction `document.querySelectorAll` qui renvoie une liste de noeud "vérifiant" le selecteur. Exemple : `document .querySelecteurAll ("p,v.maClasse")`.

Cours 66. *Cours (bilan selecteur)*

— .class
— #identifiant
— + petit frère
— g̃rand frère
— > enfants directs
— (blanc) : enfant indirect
— , : ou logique

HTTP

Cours 67. *netcat*

Le programme `netcat` permet de communiquer a travers une liaison TCP/IP. Pour attendre une liaison (en mode "serveur") lancer la commande `$netcat -l 1503`, 1503 est arbitraire (mais compris entre 1024 et 65536) c'est un numéro de port.

N'importe quel processus utilisant TCP/IP peut s'y connecter, un autre `netcat` par exemple (mais en mode "client"). Avec netcat la commande donnerais `$netcat 127.0.0.1 1503`, `127.0.0.1` est l'adresse **ip** d'un système vu par **lui même** 1503 est arbitraire mais doit correspondre au même numéro que le `netcat` "serveur".

Exercice 68. *Aliénation*

Faites communiquer deux consoles sur la même machine avec `netcat`.

Exemple en parallèle. Après la liaison, chaque message validé dans une console sera affiché sur l'autre console.

```
machin@machine$ netcat -l 1076          machin@machine$|
|
```

```
machin@machine$ netcat -l 1076          machin@machine$ netcat 127.0.0.1 1076
|                                       |
```

```
machin@machine$ netcat -l 1076          machin@machine$ netcat 127.0.0.1 1076
Bonjour|                                |
```

```
machin@machine$ netcat -l 1076          machin@machine$ netcat 127.0.0.1 1076
Bonjour                                 Bonjour
|                                       |
```

```
machin@machine$ netcat -l 1076          machin@machine$ netcat 127.0.0.1 1076
Bonjour                                 Bonjour
|                                       Salut|
```

```
machin@machine$ netcat -l 1076          machin@machine$ netcat 127.0.0.1 1076
Bonjour                                 Bonjour
Salut                                   Salut
|                                       |
```

Cours 69. *Adresse Ip*

Une liaison TCP/IP peut se faire à travers un réseau IP (utilisant le **P**rotocole **I**nternet) il faut connaître l'adresse IP du système qui exécute le processus avec lequel vous souhaitez communiquer. Le numéro de port sert à distinguer le processus parmi les autres sur un même système.

Exercice 70. *double port*

Essayer de lancer deux netcat "serveur" (`netcat -l ....`) sur le même numéro de port sur la même machine. Que se passe-t-il ? Pourquoi ?

Exercice 71. *Adresse ip*

A l'aide de la commande `$ ip a` récupérez l'adresse ip (v4) de l'interface ethernet sur la machine de la salle puis préparez-vous à la communiquer (une adresse ip version 4 est présenté par 4 nombres compris entre 0 et 255 séparés par des points par exemple `192.168.162.4` ou `127.0.0.1`). Pour vérifiez, si par exemple votre adresse est `192.168.162.4`, vous pouvez essayer `$netcat 192.168.162.4 1503`.

Exemple

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> .......
    link/loopback 00:00:00:.....
    inet 127.0.0.1/8 scope ....
.......
2: enp0s31f6: <BROADCAST,MULTICAST,UP,LOWER_UP>.......
```

```
    link/ether 18:db:f2.....
    inet 192.168.162.11/24 brd 192.168.162.255... ## l'adresse ip est 192.168.162.11
       valid_lft 404sec preferred_lft 404sec
.....
3: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500......
......
```

Cours 72. *socktet*

Généralement, un système fournit l'interface de programmation de type 'socket' pour établir une liaison TCP/IP. Une fois la liaison établie, elle est parfaitement symétrique, on peut écrire des octets dans le socket comme on le ferait dans un fichier (TCP : mode flux) ce qui aura pour effet de les envoyer dans l'ordre à l'autre socket, lesquels seront ensuite mis dans une file en attendant d'être lus, on peut faire ça des deux bouts (sur les deux sockets de la liaison).

`netcat` ne fait que relier sa sortie standard aux octets reçus venant de la liaison et transmet les octets reçus par son entrée standard vers la liaison.

Exercice 73. *SocketPython*

Voici un programme `python3` qui tente de se connecter à un couple (ip,port) (à modifier) et envoie le message "Bonjour" puis quitte. Essayez modifier le programme afin de recevoir le message avec `$netcat -l 1076`.

```
import socket                                                 # bibliotheque socket
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # (ipv4,mode TCP)
socket.connect(("215.125.152.251", 7777))                     # (adresse-ip,num-port)
socket.send(b"Bonjour")                                       # envoie liste octets
socket.close()                                                # ferme la liaison
```

`connect` met le programme en pause au plus quelque secondes tant que la connexion n'est pas établie (évitez le nom `socket.py`).

Cours 74. *socket serveur Python*

Coté serveur c'est différent, il faut un socket qui écoute les tentatives de liaisons, et ensuite il faut un socket par liaison acceptée.

```
import socket                                                  # bibliotheque socket
socEcoute = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # (ipv4,mode TCP)
socEcout.bind(("0.0.0.0",12345))                               # 0.0.0.0 (accepte tout)
socEcout.listen()                                              # pres a recevoir
socLie,adresseLointaine=socEcout.accept()                      # produit socket pour la
                                                               # première connexion reçu
c=socLie.recv(10)                                              # lit 10 prochains octets
                                                               #  si len(c)=0 l'autre bout
                                                               # a fermé
```

`.listen` et `.recv` attendent un évènement externe, ce qui qui met en pause le programme.

Cours 75. *Communication*

Mettre en place une communcation entre deux processus python.

Cours 76. *Web*

On peut découper une URL comme `http://a.michelizza.free.fr/pmwiki.php` en trois parties : `http` `a.michelizza.free.fr` et puis le reste `/pmwiki.php`.

La seconde partie `a.michelizza.free.fr` représente une adresse ip et par défaut le numéro de port est par défaut 80 pour http. On aurait pu écrire `a.michelizza.free.fr:80`, Pour obtenir l'adresse ip associée : `$ host a.michelizza.free.fr`

Nous laissons de coté le `https` pour plus tard.

Pour récupérer la ressource associée à l'URL on peut utiliser une requête GET avec netcat (pour ne pas perdre trop de temps à taper la requête on utilise la commande `echo` et `|` qui redirigera la sortie standard `echo` vers l'entrée de la commande `netcat`.

```
$ host a.michelizza.free.fr
..... has adress 212.27.34.12
$echo -e "GET /pmwiki.php HTTP/1.1\nHost: a.michelizza.free.fr\n\n" | netcat -C 212.27.34.12 80
HTTP/1.1 200 OK
Date: Fri, 13 Feb 2026 17:19:47 GMT
Server: Apache/ProXad [Jan 23 2019 20:05:46]
Cache-Control: no-store, no-cache, must-revalidate
Expires: Tue, 01 Jan 2002 00:00:00 GMT
X-Powered-By: PHP/4.4.3-dev
Connection: close
Content-Type: text/html; charset=ISO-8859-1;

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
....
....
```

Une requête GET minimale HTTP version 1.1 est donc de la forme.

```
GET / HTTP/1.1
Host: mon.nom.de.domaine
```

Un saut de lignes doit être représenté par la succession des deux octets 0x0D 0x0A (option -C). Ici la fin de la requête est marquée par deux sauts de lignes successifs (une ligne blanche donc). On peut ajouter des informations supplémentaires avant la ligne vide.

Remarquez que la réponse : 1. est précédée d'un entête terminé par un saut de ligne. 2 : La fin de la requête et marquée par la terminaison de la connexion (envoyé par le serveur).

La réponse est un document html (vieille version).

Exercice 77. *Manuel*

Récupérer la page web associée à la ressource
`http://a.michelizza.free.fr/pmwiki.php?n=TutoCFrench.TutoCFrench`
dans un fichier `page.html` et affichez la correctement (sans problème d'encodage) dans un navigateur, (vous pouvez la modifier un peu).

Cours 78. *nom de domaine*

On peut remplacer le domaine par une adresse ip suivit par un numéro de port, par exemple
`http://148.123.100.56:1076/mapage.html`

Exercice 79. *Requête du navigateur*

Donnez un exemple d'une requête GET envoyé par votre navigateur (utilisez netcat pour la récupérer).

Cours 80. *Serveur web*

Le fonctionnement du protocole HTTP/1.1 est le suivant à une requête (par exemple GET ou autre) on y répond en deux morceaux : un entete, et un contenu (la taille du contenu est donné dans l'entête) puis fin de l'échange. Les versions recentes de HTTP font la même chose mais avec des améliorations de performances et de nouveaux types d'entêtes (même sur d'autre protocole que TPC/IP).

Écrire un serveur web c'est écrire un programme qui répond à des requêtes HTTP. Le plus souvent par une page.

La commande `python3 -m http.server 9000` lance un serveur web dans le dossier courant écoutant sur le port 9000 dont le travail est simplement de renvoyer les fichiers du dossier et des sous-dossiers.

Exercice 81. *Site web local*

Publiez (et vérifiez) sur votre machine en salle, sur le port `12345`, une page html nommée `mapageweb.html` (ne changez pas le nom) sur laquelle vous aurez indiqué un texte de votre choix. Ensuite consultez les pages web des autres machine (à l'aide des adresses publiés au tableau), qui ont les mêmes trois premiers numéros d'adresse ip, le dernier numéro change selon la machine de la salle.

Cours 82. *Syntaxe basique HTTP*

Une requête http se fait en deux phases : Requête - Réponse. Il existe différentes types de requêtes (GET en est un) mais la requête suit la syntaxe suivante (# ne fait pas partie de la syntaxe).

```
$NOMREQUETE $CHAINE HTTP/$VERSION      # exemple  $NOMREQUETE<--GET $VERSION<--1.1
$ENTETE                                # $ENTETE pas de ligne blanche
                                       # ligne blanche (fin d'entete)
$CORPS                                 # $CORPS (potentiellement vide)j
```

Et la réponse est sous cette forme :

```
HTTP/$VERSION $CODE $MOT    # exemple $MOT<-- OK $VERSION<--1.1 $CODE<--200
$ENTETE                     # $ENTETE (pas de ligne blanche)

$CORPS
```

Voici quelques codes souvent répondus : `200 OK` | `404 Not Found` | `400 Bad Request`

Comme exemple de ligne d'entête, `Connection: close` indique que la fin de la réponse HTTP est indiqué par la fermeture de la connexion TCP/IP.

Exercice 83. *TP 4 5 Serveur web simplifié*

En vous inspirant du serveur `http.server` de python, écrivez programme monServeur.py qui, lorsqu'il est lancé se comporte comme un (quasi-) serveur web écoutant sur le port 12345 n'appliquant qu'une version simplifiée du protocole HTTP puisqu'il ne comprend que la requête GET du protocole HTTP/1.1. Il peut ne connaître qu'une page (de votre choix) à l'adresse `/` et doit la renvoyer (vous pouvez aussi l'améliorer en ajoutant d'autres aspects du protocole). Dans tous les autres cas il renvoie 404 ou 402.

Variables

Cours 84. *CGI*

Un serveur utilisant le protocole cgi (common gateway interface) permet de construire une réponse à une requête HTTP à la volée à partir d'un fichier exécutable.

- Les fichiers exécutables se trouvent dans un sous-dossier nommé `cgi-bin` à l'interieur du dossier d'exécution du serveur.
- La fin de la réponse est contruite par l'exécutable et transmise sur sa sortie standard. L'autre partie (début de l'entête) est complété par le serveur.
- Le contenu transmis commence à partir de l'entête (on peut donc indiquer l'encodage et le contenu, par exemple `content-type: text/html` si c'est une page web).
- L'entête de la requête (GET,POST,PUT,...) est transmis à l'exécutable par les variables d'environnement. Le corps de l'entête (s'il y en a un) est transmis par l'entrée standard.

Exercice 85. *premier cgi*

La commande `python3 -m http.server --cgi 12345` lance un serveur http en mode CGI (sur le port 12345). Créer un exécutable `/cgi-bin/bonjour` qui affiche la page bonjour.

Exercice 86. *date*

Créer une page `/cgi-bin/date` qui affiche la date du serveur.

Cours 87. *Formulaire GET*

Un formulaire get d'une page web est un ensemble de champs permettant de recueillir des valeurs entrées par un utilisateur munit d'un "bouton" d'envoi. Lorsque l'utilisateur clique sur envoi, ces valeurs sont transmises dans la partie URL de la requête. Si la réponse est une page web elle est elle affichée.

Voici un exemple minimal d'un formulaire GET (c'est à dire qui transmet des valeurs avec une requête GET).

```
<form action="http://....." method=GET>
<input type="text" name="monIdentifiant1">
<input type="submit">
</form>
```

Exercice 88. *Exercice*

Dans quel format les valeurs du formulaire sont-elles transmises au serveur (On peut utiliser les outils du navigateur pour consulter les contenus d'une requete) ?

Cours 89. *CGI*

Les valeurs des champs de l'url sont transmises dans les variables d'environnement du processus exécuté.

Exercice 90. *Message*

Créer une page `/cgi-bin/message` qui affiche un message reçu en paramètre. exemple d'url `http://127.0.0.1:12345/cgi-bin/message?mes=bonjour+tout+le+monde`. Vous pouvez créer une page `/formulaire.h` avec un formulaire.

Cours 91. *Cours*

Dans un formulaire, plusieurs valeurs de l'attribut `type` sont disponibles pour la balise `input`, l'attribut `value` est aussi disponible pour une valeur par défaut.

- text (pour du texte)
- textarea (grande zone de texte)
- password (pour du texte masqué)
- checkbox (case à cocher)
- number (un nombre)
- url (une url)
- tel (numéro de téléphone)
- email (adresses emails)
- date (date sans heure)
- time (heure sans date)
- date-time-local (époque, sans secondes)
- hidden (valeur constant invisible)
- radio (case à cocher, une seule parmie plusieurs utilisant le même 'name' mais avec différentes values)

Exercice 92. *Caractère spéciaux spécial url*

Comment plusieurs valeurs sont-elles envoyées avec un formulaire ?

Certains caractères de l'url servent à structurer l'url les voici : `! # $ & + ' ( ) * + , / : ; = ? [ ] %` Comment le navigateur procède-t-il pour les transmettre ?

Cours 93. *Post*

Il existe une autre requête http que l'on appelle la requête post. La convention http indique qu'une requête get ne doit pas modifier quoi que ce soit sur le serveur (on dit qu'elle est invariante) alors qu'une requete POST le peut.

Exercice 94. *formulaire POST*

A votre avis, que doit-on modifier dans un formulaire pour que l'action associée soit l'envoie d'une requête post ?

Exercice 95. *Forme d'une requête post*

Quelle est la forme d'une requête post d'un formulaire ? Repérer `content-type` et `content-length`.

Cours 96. *POST cgi*

Le corps d'une requête en CGI est transmise par l'entrée standard. On peut connaître la taille du corps de la requête avec la valeur du champs `content-length` (contenu dans l'entete de la requête).

Exercice 97. *TP 5 Base de données simple*

Concevoir et exécuter un serveur web avec une seule adresse : `/cgi-bin/bddsimple`. La page (ou script) `bddsimple` doit contenir un formulaire texte que l'utilisateur peut soumettre. Le message soumis est enregistré et doit apparaître ensuite à la suite du formulaire. Les messages peuvent ainsi s'accumuler. Vous pouvez sauvegarder les messages dans un fichier texte (attention au format html).

Base de données (relationnelle)

Cours 98. *Gestionnaire de base de données*

On peut organiser des données en utilisant un gestionnaire de base de données comme par exemple SQLite (version 3). En sqlite une base complète est entièrement écrite dans un seul fichier, avec d'autres gestionnaires

(comme postgresql ou mysql) la base de données est destinée à être modifié à travers un serveur TCP. SQLite est fournit avec la commande `sqlite3`, un interpréteur interactif.

Exercice 99. *Sqlite ?*

Pour vérifier que sqlite est disponible sur votre système tapez les commandes suivantes dans un terminal. Le second argument (`maBase.sqlite3`) de la commande `sqlite3` indique l'emplacement où se trouve la base de données sur laquelle on veut travailler, si le fichier n'existe pas il sera eventuellement créé. Les différentes commandes entrées dans l'interpréteur se raporteront à cette base. La seconde ligne quitte l'interpréteur.

```
machin@machine$ sqlite3 maBase.sqlite3
SQLITE version .......
Enter ".help" .....
sqlite>.quit
```

Cours 100. *SQL*

SQLite utilise le modèle dit relationnel de données. Dans ce modèle les données SONT un ensemble de tableaux à deux dimensions (avec lignes/entrées et colonnes/champs) nommés tables. Le langage informatique SQL est conçu pour traiter des données suivant ce modèle. SQLite utilise une variante de SQL.

Une commande SQL se termine par un point virgule. Un commentaire de fin de ligne commence par `--`.

Exercice 101. *SQLite*

A un emplacement `xxx.sqlite3` (de votre choix), à l'aide de l'interpréteur `sqlite3`, lancer les commandes (ou requêtes) SQL suivantes (on peut reprendre l'historique avec 'flèche du haut').

```
sqlite> CREATE TABLE "maTable" ("nom","prenom","age");   -- colonnes : "nom","prenom","age"
sqlite> INSERT INTO "maTable" VALUES ('SEGUR','Sophie',20);      --ajoute 1 ligne complete
sqlite> INSERT INTO "maTable"("age",nom) VALUES (34,'FRACASSE'); --ajoute 1 ligne dont une case NULL
sqlite> INSERT INTO "maTable" VALUES ('O''BRIAN','Honoré',30);   --'' echappe un apostrophe
sqlite> SELECT * FROM "maTable";                                  --affiche toute la table
```

Cours 102. *CREATE,INSERT,SELECT*

La ligne `CREATE...` crée une nouvelle table nommée `maTable`, qui contient trois colonnes (ou champs/field) nommées `nom prenom age` . Au départ la table ne contient aucune ligne (ou entrée/entry). Les commandes suivante ajoute chacune une ligne dans la table `maTable` en indiquant les valeurs associées à certaines colonnes. La dernière commande affiche (`*` : toutes) la table `maTable` (sans rien modifier).

Une liste d'octets entre guillemets anglais représente un identifiant. Les guillemets sont quelquefois optionnels mais permettent de lever des ambiguités. Un encadrement entre apostrophes est une chaîne de caractère. Doubler un appostrophe permet de l'échapper, par exemple `O'BRIAN` peut s'écrire `'O''BRIAN'`.

Exercice 103. *Nom Prenom*

Ajouter une personne nommée "Aristide FIL'SELLE" à la table précédente (sans indiquer son âge).

Cours 104. *Toutes les tables*

SQLite fournit une table 'virtuelle' nommée `sqlite_master` dont chaque ligne est un élément de la base (il peut y avoir d'autres éléments que des tables).

Exercice 105. *Anormal sup*

Donner la liste des tables de la base `anormalSup.sqlite3` (voir site du cours).

Cours 106. *sqlitebrowser*

La commande `$ sqlitebrower anormalSup.sqlite3` ouvre une fenêtre graphique permettant d'explorer et modifier la base. On peut entre autres, consulter la liste de tables, modifier des tables, les afficher, exécuter des requêtes sql. Les modifications apportées sont prises en compte lorsqu'on clique sur le bouton "enregistrer les modifications".

Exercice 107. *Cohérence*

Vérifiez la cohérence entre la liste des tables trouvée à l'exercice précédent et ce qu'indique sqlitebrower.

Cours 108. *SELECT*

La commande/requête `SELECT` ne modifie pas la base. Elle possède plusieurs options/arguments permettant de faire plus que d'afficher une table existante. Par exemple, après le mot SELECT, on peut indiquer les noms des colonnes/champs dont on souhaite afficher les contenus (`*` pour tous les conserver).

Exercice 109. *Choisir*

Quel est le résultat de la requête `SELECT "dateConst","modele" FROM "Navire";` sur la table `"Navire"` suivante :

modele	matricule	dateConstr
Ovni37	SA-104	01/02/2003
Antares25	PO-32	14/08/1996
Mojito650	CH-435	31/07/1990
Aventura37	NA-23	21/12/2001
First29	BR-4567	01/08/2007

Cours 110. *instruction sqlite3*

En plus des commandes en sql, l'interpreteur `sqlite3` reconnaît des commandes qui lui sont propre comme par exemple `.quit`.

On peut utiliser l'interpréteur non interactivement dans un shell :
`machin@machine$ sqlite3 emplacement commande1 commande2 commande3`.

Voici différentes commandes en plus de `.quit`.

- `.help` donne des informations sur ces instructions
- `.quit` quitte l'interpreteur
- `.read emplacement` execute les commandes contenues dans le fichier 'emplacement'.
- `.dump` exporte la base courante sous la forme d'une liste de requêtes sql.
- `.mode table` change le format de sortie (d'instruction comme SELECT) en mode `table`. Voici d'autres formats : `html`, `table`, texte agréable `insert maTable`, sous la forme d'instructions sql pour construire la table `maTable`, `json`, `csv`.

Exercice 111. *Extraire table*

Construire une base qui contient une seule table `Etu` contenant la liste des noms et prénoms des étudiants de la base `anormalSup.sqlite3`.

Cours 112. *WHERE*

Dans une requête SELECT la clause WHERE permet de ne selectionner que certaines lignes vérifiant une expression booleenne.

Exercice 113. *Selectionner*

Quel est le résultat de la requête `SELECT * FROM "personne" WHERE "age"<37;` sur la table `personne` suivante :

nom	prenom	age
Guyot	Jacqueline	17
Fontaine	Nicole	37
Petit	Véronique	52
Dubois	Chantal	26
Durand	Sylvie	4

Cours 114. *Interaction programme*

Sqlite est essentiellement une bibliothèque C permettant d'appliquer des commandes SQL sur un fichier (au format sqlite3). Le module python `sqlite3` permet d'utiliser cette bibliothèque.

```
con=sqlite3.connect("anormalSup.sqlite3")
cur=con.cursor()
cur.execute("""CREATE TABLE "bonjour" ("test");"""))
con.commit()                                          #pour sauvegarder modifications
con.close()
```

Exercice 115. *Alachaine*

Construire une base `maBase.sqlite3` ayant 100 tables nommées `table1` `table2` ...`table100`.

Cours 116. *Valeurs*

Les 5 seuls types de valeurs possibles de sqlite3 sont : integer (signé sur 64bits,int en python), real (IEEE754 sur 64 bits,float en python, double en C), texte (chaine unicode, entre " en sql, str en python), blob (bytes en python, par exemple x'0AF3' en sql), null (None en python, NULL en sql).

Dans Sqlite, si un nombre entier ne peut pas être représenté en 64 bit signé il est approché par une valeur en IEEE754 sur 64 bit.

La méthode `execute` peut prendre un `tuple` comme second argument, dont les valeurs remplaceront les caractères '?' dans la commande sql.

`execute("""INSERT INTO "maTable" VALUE (1,'bonjour');""")` peut être remplacé par l'instruction suivante (plus sûre)

`execute("""INSERT INTO "maTable" VALUE (?,?);""",(1,'bonjour');`.

Lorsque le premier argument (commande SQL) de la méthode `execute` renvoie une table (comme le fait SELECT) alors la méthode `fetchall()` du resultat renvoie la liste des lignes de la table sous la forme d'une `list` de `tuple`.

Exercice 117. *Moyenne*

Dans la liste des étudiants de `anormalSup.sqlite`, afficher la liste des étudiants au dessus de la moyenne d'âge.

Cours 118. *Doc*

On peut consulter `https://www.sqlite.org/lang_select.html` pour plus d'informations sur la commande SELECT.

Exercice 119. *Order*

Quel est le résultat de la requête `SELECT * FROM "client" ORDER BY "nom" ASC, "prenom" DESC;` sur la table `client` suivante :

nom	prenom	dateEnregistrement
Robbe	Jean	01/02/2003
Dupont	Anne	04/05/2006
Cash	Jonny	09/07/2013
Dupont	Pierre	01/11/2011
Dentz	Charles	01/08/2007

Cours 120. *Produit cartésien*

Lorsque plusieurs tables sont indiquées après `from` la table résultante est comme un produit cartésien des lignes des tables.

Exercice 121. *Jointure*

Quel est le résultat de la requête `SELECT "figure"."forme", x FROM "position","figure";` Voici les tables `position` et `figure` :

`position`

x	y
1	3
1	2
2	4

`figure`

forme	couleur
carre	rouge
cercle	jaune

Exercice 122. *Ecrasement*

Quel est le résultat de la requête `SELECT AVG(note),matiere FROM Note GROUP BY matiere;`

Voici la table `Note` :

note	matiere
14	sport
16	math
10	philo
15	philo
18	math
5	philo

Cours 123. *Bilan SELECT*

Le schéma de select est
`WITH ... SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ....` seule la clause select est obligatoire. Les clauses sont traitées dans cet ordre : with,from,where,group by, having, order by, limit,select.

On peut utiliser des sous-requête comme des tables (temporaires). `SELECT .... FROM table, (SELECT ....) AS tempTable WHERE ... ;`

On peut demander de supprimer les doublons

`SELECT DISTINCT ....`

On peut joindre des résultats de requête avec `UNION` exemple : `SELECT ..... UNION SELECT .....`

On peut placer le résultat de SELECT dans une nouvelle table.

Les éléments de la clause from ou with peuvent aussi être des requêtes select (requêtes imbriquées).

Voir

Cours 124. *En Plus Table et sqlite*

Dans un fichier .sqlite3 on peut placer un commentaire multiligne entre `/*` et `*/`

Pour ajouter ou supprimer des tables : CREATE TABLE, DROP TABLE.

Pour modifier les valeurs d'une table UPDATE, ajouter une ligne INSERT INTO, supprimer des lignes DELETE FROM.

Pour plus d'information sur sqlite voir `https://sqlite.org/` et plus particulièrement `https://sqlite.org/lang.html` pour sa syntaxe.

Exercice 125. *TP 6 Creation Requête*

L'école Anormalsup possède une base de données (disponible sur le site du cours `anormalSup.sqlite3`) pour gérer les notes de ses élèves. Voici la liste des tables avec leurs champs :

- `Etudiant(nEtu,nom,pren,age)` liste des étudiants. `nEtu` représente le numéro d'étudiant. Seul `age` est un champs numérique.
- `Groupe(groupe,nEtu)` relation entre les étudiants et les classes (ou groupes). Chaque étudiant n'appartient qu'à une seule classe.
- `Mail (nEtu,mailP,mailS)` Liste des adresses mails des étudiants. Un mail principal, un mail secondaire.
- `Cours (nom,matiere)` Liste des cours chaque cours est associé, à une matière.
- `Coeff (cours,classe,coef)` Qui indique quelle classe (groupe) suit quel cours. Pour chaque classe et pour chaque cours `coeff` (un nombre) correspond au coefficient du cours dans la note global de chaque étudiant de la classe.
- `Examen (examen,cours,pointsMax)` La table `Examen` liste des examens donnés. Chaque examen n'est associé qu'à un seul cours. Chaque examen a été donné à l'ensemble des classes qui suivent le cours. `pointMax` est le nombre de points qu'il fallait obtenir à l'examen pour avoir 20/20.
- `Note (nEtu,examen,point)` : Chaque ligne indique le nombre de point qu'à eu l'étudiant à l'examen, il est forcément inférieur ou égal à `pointsMax`.

Seul les champs `age`, `coef` `pointMax` et `point` sont des champs numériques. Tous les autres sont des champs de texte.

Pour chaque demande suivante écrire la requête `sql` qui permet d'obtenir le résultat demandé.

1. Afficher la liste des élèves (nom,prenom)
2. Afficher la liste des étudiants qui ont moins de 18 ans.
3. Afficher la liste de toutes les matières, une seule ligne par matière.
4. Afficher (sur une colonne) la liste de toutes les adresses mails.

5. Afficher la table Note en remplaçant la note par un nombre entre 0 et 1 (0/34-> 0 12/24->0.5 20/20-> 1).
6. Afficher, pour chaque classe , le nombre d'élèves.
7. Afficher, pour chaque couple (classe,cours) la moyenne de la classe à ce cours.
8. Afficher une liste qui pour chaque étudiant (nom,prenom) donne la liste des cours qu'il suit.
9. Affiche pour chaque étudiant (nom,prenom) sa moyenne totale. (pour simplifier la requete vous pouvez créer et utiliser des tables intermédiaires/temporaires).

Illustration :
Voici une requête qui affiche la liste des noms des étudiant associé à leur adresse courriel.

```
SELECT "nom","mailP" FROM Etudiant,Mail WHERE Mail.nEtu=Etudiant.nEtu;
```