

Créer un serveur HTML+Javascript avec Arduino et afficher un canvas (zone de dessin) dans le navigateur client.



Ateliers Arduino

par X. HINAULT
www.mon-club-elec.fr



Tous droits réservés – 2012.

Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : support@mon-club-elec.fr

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : support@mon-club-elec.fr

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : support@mon-club-elec.fr

Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.

En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !

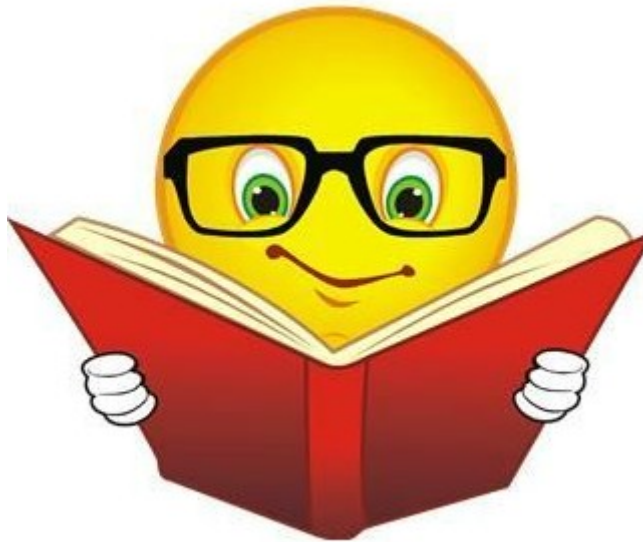
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- d'apprendre à écrire un programme javascript inséré dans une page HTML
- d'apprendre à écrire un programme javascript utilisant un canvas
- d'apprendre à écrire un programme Arduino de serveur de page HTML + Javascript

... afin d'être en mesure de créer un serveur Arduino réalisant un affichage graphique simple dans le navigateur client !



Prêt ? C'est parti !

Pratique :

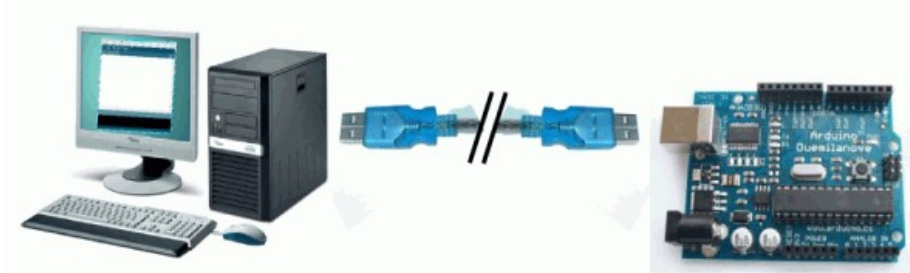
Les codes de cet atelier sont disponibles ici :

http://www.mon-club-elec.fr/mes_downloads/tutos_arduino/12b3.atelier_arduino_ethernet_tcp_javascript_canva.tar.gz

2. Matériel nécessaire pour les ateliers Arduino

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

De l'espace de développement Arduino

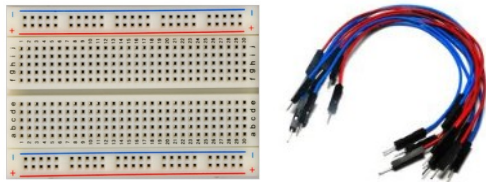


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : <http://www.arduino.cc/>)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : <http://shop.snootlab.com/> ou <http://www.gotronic.fr/>

Du nécessaire pour réaliser des montages sans soudure

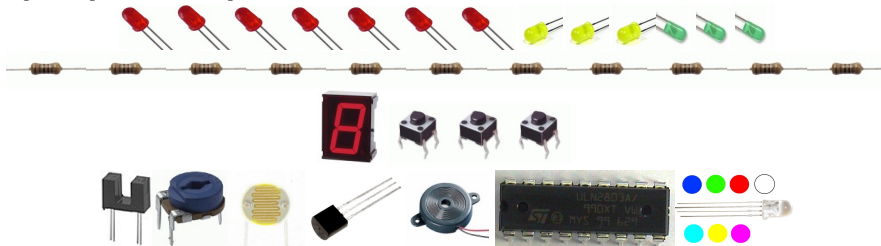


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : <http://www.gotronic.fr/>

De quelques composants de base



Pour vous simplifier la vie, nous avons négocié ce kit pour vous !

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire

<http://www.gotronic.fr/> avec le code express **701710**

GO TRONIC
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS

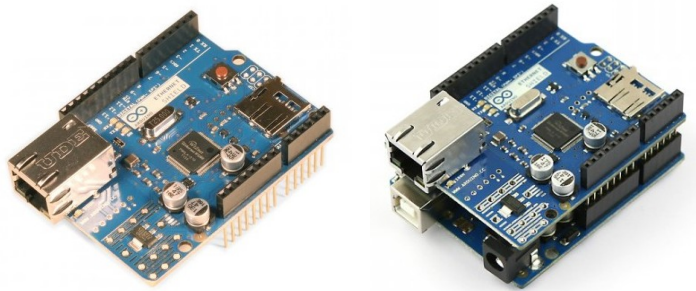
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

3. Matériel spécifique nécessaire pour cet atelier

Pour cet atelier vous aurez besoin également :

D'une carte d'extension (shield) Ethernet



La carte d'extension (ou shield) ethernet Arduino est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser Arduino sur un réseau ethernet local voire même sur internet.

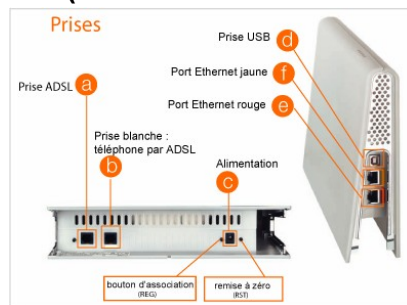
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4) pour communiquer avec Arduino.

Ce shield intègre également un emplacement pour **carte mémoire SD** pour des stockage de données ou de pages HTML locales.

Ne pas confondre ce shield avec la carte UNO Ethernet qui est une variante d'une carte UNO avec ethernet intégré.

disponible chez : <http://snootlab.com> | 33€ environ

D'un routeur Ethernet (ou d'une « box » internet)



Le routeur est un élément central du réseau qui permet de réaliser simplement un réseau local avec plusieurs postes. Ce routeur devra être de type Ethernet (réseau par fil) : si votre routeur supporte aussi le wifi, tant mieux, mais ça ne vous servira à rien ici. Votre routeur devra disposer de la fonction d'attribution automatique des adresses (ou DHCP), ce qui est le cas dans la grande majorité des cas.

A noter qu'une box internet est un routeur Ethernet (associé à un modem ADSL) et pourra ici être utilisée.

Ce routeur devra disposer d'au moins une prise réseau libre RJ45.

+/- d'un switch Ethernet (si le routeur n'a pas au moins 2 prises Ethernet libres)



Si votre routeur ne dispose que d'une seule prise RJ45, il faudra probablement que vous utilisiez également un switch réseau qui est une sorte de « multiprises » RJ45.

Bien qu'il ne soit pas toujours indispensable, je vous conseille fortement de disposer d'un switch car ce n'est pas cher (on en trouve à 10€) et ça vous permettra d'ajouter facilement des postes sur votre réseau.

4. Matériel spécifique nécessaire pour cet atelier (suite)

D'un ou 2 câbles réseau RJ45



Pour connecter les éléments du réseau Ethernet entre eux, vous devrez disposer d'au moins 2 câbles réseaux RJ45 (modèle classique, pas « croisé ») :

- 1 pour connecter votre PC au routeur
- 1 pour connecter le shield Ethernet au routeur

A moins que vous ayez l'intention de mettre votre carte Arduino loin de votre poste fixe, vous pouvez utiliser des câbles courts de 1m par exemple.

Noter qu'il existe des câbles RJ45 de petite longueur sur petit enrouleur : pratiques pour réduire l'encombrement !

Conseil d'ami : ne pas hésiter à avoir quelques câbles ethernet d'avance sous le coude...

+/- de 2 blocs CPL (seulement si vous souhaitez déployer le réseau Ethernet via le réseau électrique 220V)



Les blocs CPL (technologie à courant porteur) permettent assez facilement de déployer un réseau Ethernet sur le circuit 220V domestique, avec une portée de 200m sans difficulté.

Vous aurez besoin de cet équipement si vous souhaitez créer un réseau entre Arduino + shield Ethernet et votre poste fixe dans des pièces différentes par exemple.

Cet équipement un peu plus coûteux (compter 40€ pour un bloc de qualité) n'est pas indispensable dans une première approche. Mais sachez que ça existe.

A titre indicatif : j'utilise et je conseille les blocs Delovo AvPlus 200, qui disposent d'une prise terre en façade, sont faciles à utiliser, sont robustes au quotidien et sont livrés avec un utilitaire Linux pour la configuration.

Et d'un poste fixe (PC, Mac, Netbook,...) disposant d'une carte Ethernet !



Je pense que c'est évident, mais je préfère quand même le dire... Vous avez besoin d'un poste fixe disposant d'une carte réseau Ethernet. Celui où vous lisez cette page et avec lequel vous programmez votre carte Arduino devrait faire l'affaire.

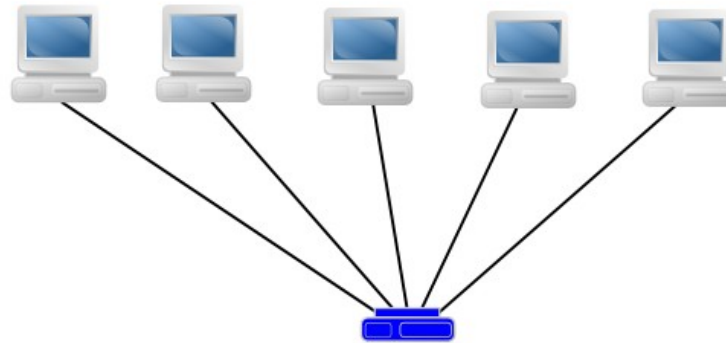
Votre poste peut-être sous Windows, Mac OS X ou Gnu/Linux, peu importe. Vous pouvez utiliser indifféremment un PC de bureau, un netbook ou un portable.

5. Rappel : structure d'un réseau local et matériel nécessaire

Structure générale

Techniquement, un réseau local type va avoir la même structure quelque soit la technique de connexion utilisée (filaire ou wifi) :

- l'un des postes va être le « meneur du jeu » : on l'appelle le routeur (en bleu sur le schéma). C'est lui qui :
 - fixe le numéro du réseau (couleur du maillot)
 - attribue les numéros individuels des postes (les numéros des joueurs), (rôle de serveur DHCP)
 - voit tous les postes
 - et distribue les messages (rôle de commutateur / switch sur le réseau local et/ou rôle de routeur si connexion au réseau extérieur)
- les joueurs sont l'ensemble des postes du réseau qui sont connectés au routeur.
- Les numéros des joueurs et du meneur sont appelés « adresse IP » et sont attribués par le routeur (mode automatique dit « DHCP »).



Le matériel de base nécessaire

- Pour constituer un réseau local filaire (le plus simple au début), on a donc besoin :
 - d'un **routeur** (ethernet ou wifi ou mixte) ou d'une box (qui est un routeur + modem)
 - de **plusieurs câbles RJ-45** pour connecter les éléments entre eux
 - d'un ou plusieurs postes à connecter sur le réseau
 - +/- d'un « multiprise » réseau, appelé switch, si le routeur n'a pas assez de connecteurs RJ-45 (Attention : un switch n'est pas un routeur... mais le routeur est un switch sur le réseau local !)



Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

Atelier Arduino : Créer un serveur HTML+Javascript avec Arduino et afficher un canvas (zone de dessin) dans le navigateur client. p.6/52

6. Un peu de vocabulaire pour avoir les idées claires

Avant de poursuivre, voici quelques définitions pour vous aider à avoir les idées claires :

Réseau :

Un ensemble de postes informatiques / électroniques reliés entre eux et capables de communiquer entre eux.

Réseau local :

Réseau constitué par les postes d'un même réseau (dans la maison ou le bureau). Le réseau local « pur » n'utilise pas d'accès vers l'extérieur, mais seulement une connexion de type Ethernet. Pour reprendre l'image du jeu, un réseau est la table de jeu avec le meneur et les joueurs autour. Comme nous allons le voir, chaque « table de jeu » va avoir un numéro qui permet d'identifier un réseau local donné.

Réseau internet ou web :

Réseau qui relie par le réseau téléphonique, et/ou fibre optique, des postes individuels et des réseaux locaux entre eux.

Ethernet :

Réseau où les postes sont reliés entre eux par fil à l'aide d'un câble spécial appelé RJ45.

Wifi :

Réseau où les postes sont reliés entre eux sans fil, par ondes radios.

Modem :

Appareil permettant de transmettre des données informatiques à l'aide du réseau téléphonique. C'est la fonction « modem » qui permet de se connecter à internet. La technologie utilisée actuellement par les modems est dite « ADSL ». Ici, cette fonction modem du routeur ne sera pas indispensable, mais pourra être utile pour certains programmes.

Routeur :

Appareil permettant aux différents postes d'un réseau Ethernet local de communiquer entre eux. Dans le jeu précédent, le routeur est le meneur de jeu. Le routeur va à la fois fixer le numéro de la table (= numéro du réseau) et le numéro de chaque joueur sur le réseau.

Noter qu'un routeur peut fonctionner soit en ethernet (fil) soit wifi (sans fil) soit les 2 (cas le plus courant des box internet actuelles)

Box internet

Appareil électronique et informatique qui permet de se connecter à internet et à créer un réseau local domestique. Bien comprendre qu'une box est à la fois un modem (connexion à internet par le réseau téléphonique) et un routeur (qui permet aux ordinateurs d'un même réseau de communiquer entre eux).

Carte Réseau (Ethernet) :

Carte d'interface électronique intégrée à un ordinateur ou un dispositif et qui permet de communiquer sur un réseau filaire Ethernet (local). Le shield Ethernet est une mini carte Réseau.

Carte wifi :

Carte d'interface électronique intégrée à un ordinateur ou un dispositif et qui permet de communiquer sur un réseau wifi sans fil (local). Le shield Ethernet est une mini carte Réseau.

Adresse IP :

Le numéro qui est attribué à chaque poste sur le réseau.

DHCP :

Fonction du routeur qui permet d'attribuer automatiquement les numéros (ou adresse IP) aux postes du réseau (les joueurs de la table...). Retenir que le routeur utilise une plage d'adresses prédéfinies, propre à chaque routeur.

IP fixe :

Se dit d'un poste du réseau dont l'adresse IP est fixée manuellement au lieu d'être attribuée par le routeur.

Serveur :

Se dit d'un poste du réseau qui répond à des requêtes en provenance d'un client. Un site internet par exemple est en fait un serveur réseau.

Client :

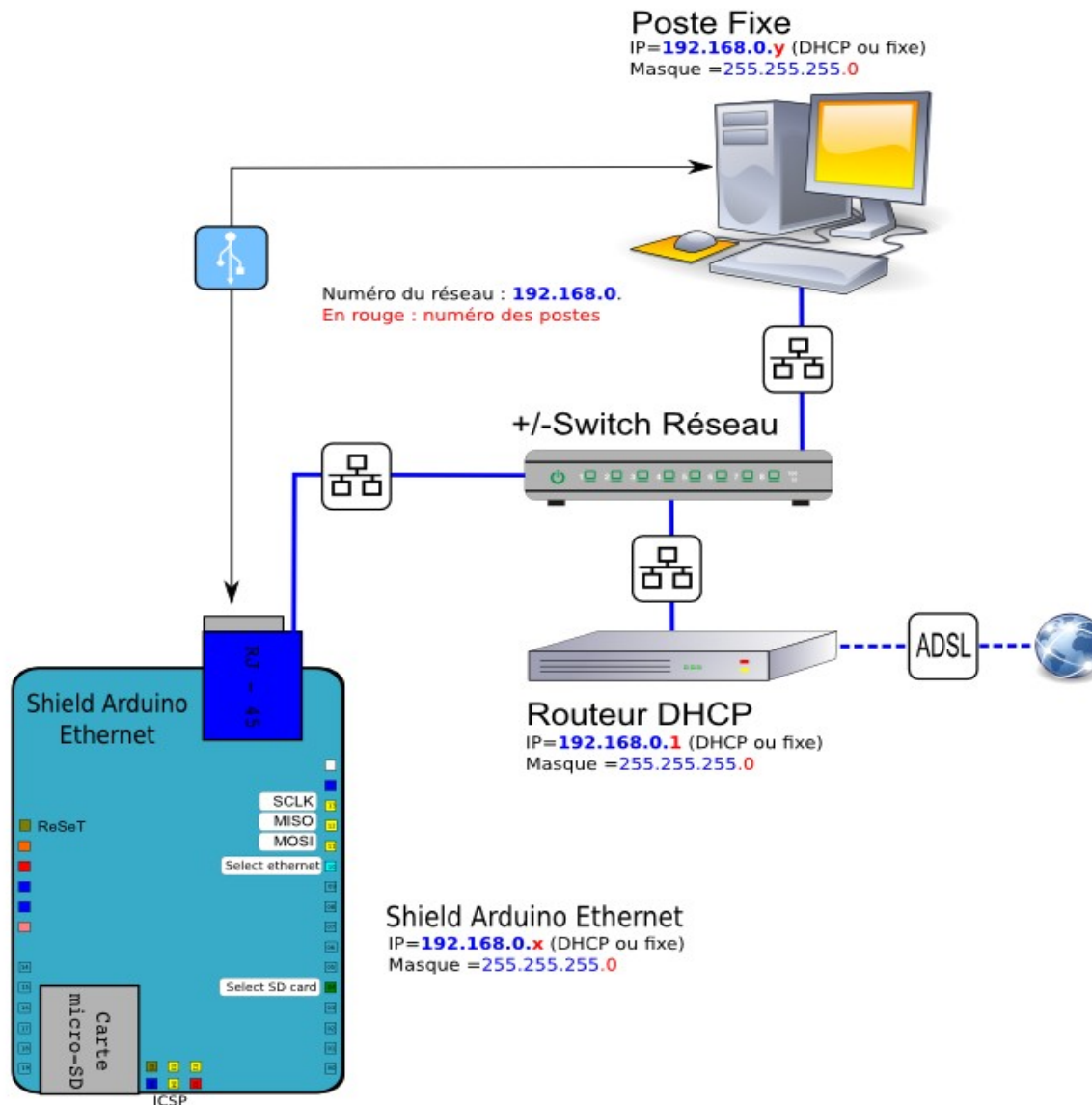
Se dit d'un poste du réseau qui envoie des requêtes (des demandes) à un serveur. Un navigateur sur un poste fixe constitue un client réseau.

Protocoles TCP/IP et UDP :

Les protocoles TCP/IP et UDP correspondent à la façon dont les postes d'un réseau communiquent entre eux : TCP/IP est le protocole de l'internet, UDP un protocole plus simple.

Si vous ne retenez pas tout pour le moment, ce n'est pas très grave. Les choses vont s'éclaircir progressivement et vous pourrez revenir sur cette page si besoin.

7. La structure du réseau que nous allons réaliser



Notre réseau utilisant Arduino va être constitué au minimum :

- d'un **routeur ethernet** (ou d'une box) fonctionnant en mode DHCP (=attribution automatique des adresses) avec au moins 1 prise ethernet RJ45 libre
- +/- d'un **switch réseau** (=«multiprise » réseau) si le routeur ne dispose que d'une prise ethernet RJ45
- d'un **poste fixe**, le pc sur lequel vous travaillez, connecté au routeur directement au routeur ou sur le switch avec un câble ethernet RJ45
- d'un **couple « carte Arduino + shield Ethernet »** connecté également directement au routeur ou sur le switch avec un câble ethernet RJ45

Dans un premier temps, le routeur n'a pas besoin d'être connecté à Internet.

Si il y a plus d'éléments sur votre réseau, cela n'a aucune importance, mais dans un premier temps, mieux vaut faire simple.

Remarquer que le couple « Arduino/shield Ethernet) est connecté au PC fixe de 2 façons :

- par USB d'une part
- et par ethernet d'autre part

Ceci est très pratique en phase de test et de mise au point, mais une fois la programmation terminée, on pourra bien sûr déconnecter le câble USB.

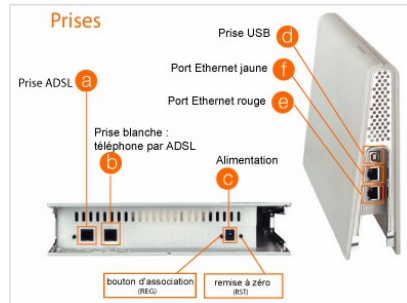
La signification des numéros (adresses IP) indiqués sur ce schéma seront expliqués par la suite.

8. Les éléments du réseau local que nous allons utiliser

Bien, à présent, trêve de « blabla », passons aux choses concrètes. Voyons à quoi va ressembler notre réseau.

Le premier élément du réseau : le routeur (ou une box).

Je me place ici dans le cas de figure courant de nos jours où vous disposez d'une box internet qui assure la double fonction de routeur et de modem. Les box intègrent la fonction DHCP d'attribution automatique des adresses IP. Votre box est le premier élément de votre réseau.



Si vous n'avez pas de box, vous devrez au moins disposer d'un routeur ethernet qui supporte la fonction DHCP, ce qui est le cas de la plupart des routeurs récents.

Noter que si vous utilisez un routeur ou une box non connecté à internet, ce n'est pas grave : vous n'aurez pas besoin de l'accès à internet pour la majorité des programmes que je vous propose.

Sur votre routeur, vous devez disposer d'au moins une prise ethernet RJ45, et idéalement 2.

+/- Élément complémentaire du routeur : le switch Ethernet

Si vous ne disposez que d'une prise Ethernet sur le routeur (c'est parfois le cas sur les box internet), vous devrez également utiliser un switch réseau qui est une sorte de multi-prises RJ45.



Le second élément de votre réseau : le poste fixe où vous travaillez disposant d'une carte réseau

Le second élément du réseau est le poste fixe où vous travaillez. Ce poste doit disposer d'une interface Ethernet filaire (ou carte réseau), L'autre possibilité est que le poste fixe se connecte au routeur par wifi, mais ce cas de figure n'est pas idéal ici car cela pourrait entraîner certains problèmes de connexion. Donc dans un premier temps au moins, utiliser de préférence une connexion Ethernet pour votre réseau.

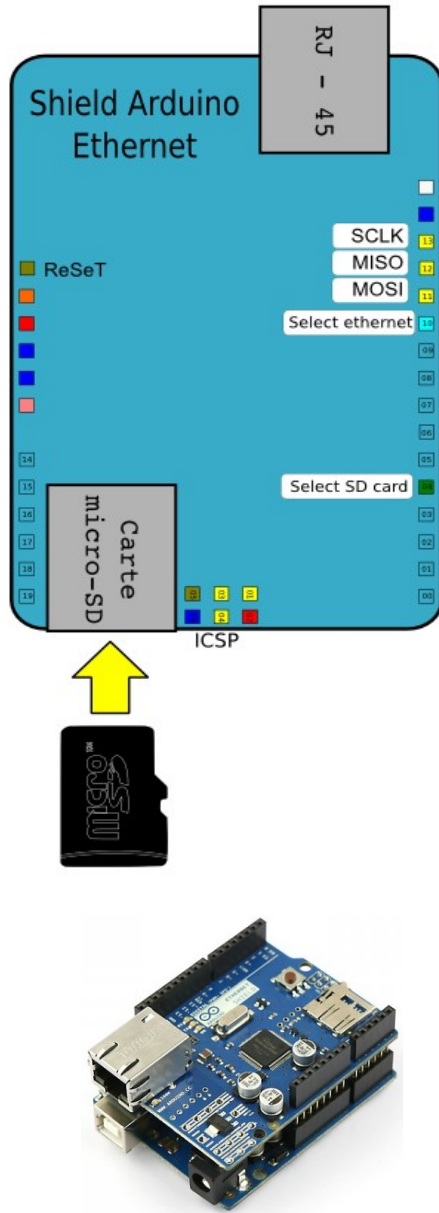


Le troisième élément de votre réseau : le couple « carte Arduino + shield Ethernet »

Le troisième élément du réseau est bien évidemment le shield Ethernet qui sera tout simplement enfiché broche à broche sur la carte Arduino



9. Le shield Ethernet : description et principe d'utilisation



Description

- Le module Ethernet Arduino permet à une carte Arduino de se connecter à un réseau Ethernet filaire ou même à internet.
- Ce module intègre également un emplacement pour une carte mémoire micro-SD, pouvant stocker plusieurs Go de données !
- Ce module est basé sur le circuit intégré [Wiznet W5100](#). Le Wiznet W5100 fournit une pile réseau (IP) capable à la fois de **TCP et UDP**. Il supporte jusqu'à quatre connexions simultanées.
- Il suffit d'utiliser la **bibliothèque Ethernet** pour écrire des programmes qui se connectent à un réseau ou à internet en utilisant ce module.

Brochage

- Ce shield communique avec la carte Arduino en utilisant une communication SPI (voir atelier dédié pour plus de détails). Cette communication utilise les 3 broches suivantes 11 (MOSI), 12 (MISO) et 13 (CLK). *Noter que la connexion de ces broches se fait par le connecteur ICSP de la carte Arduino.*
- La sélection de l'étage utilisé (carte SD ou Ethernet) se fait à l'aide de 2 broches de sélection :
 - la broche 10 pour sélectionner l'étage Ethernet
 - la broche 4 pour sélectionner la carte mémoire SD

La gestion des broches sera assurée automatiquement par les bibliothèques.

- Toutes les autres broches de la carte Arduino restent disponibles pour d'autres utilisations.

Principe d'utilisation

- Le module ethernet se connecte « broche à broche » sur une carte Arduino grâce à ses longues broches qui dépassent du circuit imprimé. On dit que le shield est « stackable » !
- Ainsi le brochage de la carte Arduino n'est pas modifié et permet d'enfiler un autre module par dessus et laisse l'accès aux broches de la carte Arduino.

10. Monter le réseau utilisant le shield Ethernet Arduino

- A faire hors tension dans la mesure du possible... Connecter le câble RJ 45 à la carte réseau du poste fixe :



- Connecter le câble RJ45 au routeur (ou à la box) :



- Mettez le shield Ethernet en place sur la carte Arduino :



- Puis connecter le shield Arduino au routeur à l'aide d'un câble RJ-45 :



- Connecter enfin le câble USB entre l'ordinateur et la carte Arduino.



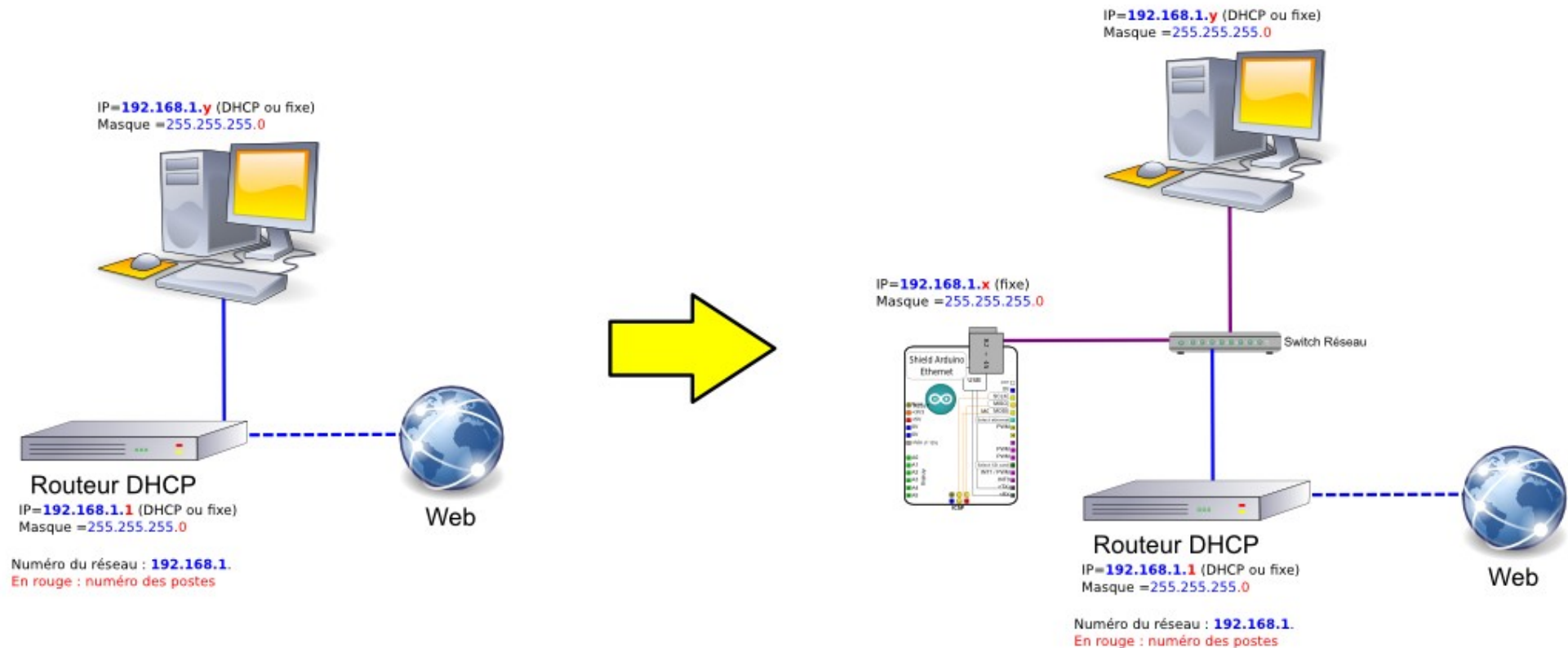
Dans notre cas, on connecte le couple Arduino + shield Ethernet à la fois en USB et en Ethernet au poste fixe, ceci uniquement à des fins didactiques et pour faciliter les développements.

En situation réelle, évidemment, le couple Arduino + shield Ethernet pourra être utilisé à distance du poste fixe, pourvu qu'il soit connecté au réseau Ethernet.

11. Monter le réseau utilisant le shield Ethernet Arduino sur un réseau avec « box » existant

Remarque :

Si votre poste fixe est déjà connecté à votre box internet, la manip' à réaliser est simple : il suffit de débrancher le câble ethernet de votre poste fixe et de le brancher sur le switch réseau. Ensuite, connecter un câble entre le switch réseau et votre PC. Puis un second câble entre le switch réseau et le shield Ethernet enfiché sur la carte Arduino. C'est tout.



12. Rappel : Structure type d'une page HTML simple et écrire une première page HTML

Quelques balises essentielles

- De ce qu'on a dit précédemment, vous connaissez la balise **
** qui correspond au saut de ligne.
- La plupart des autres balises fonctionnent 2 par 2 avec une balise de début **<balise>** et une balise de fin **</balise>**
- la première balise à connaître est la balise **<html> </html>** : ces balises signalent le début et la fin du contenu de la page html. Tout ce qui sera compris entre ces 2 balises sera interprété comme de l'html.
- Une balise utile est celle qui délimite les commentaires : **<!-- -->**
- Toute page html comporte une obligatoirement entête contenant diverses informations et paramétrages de la page. L'**entête** est délimité par les balises **<head> </head>**
- Une balise utile au sein de l'entête est celle du titre de la page délimité par les balises **<title> </title>**
- Toute page html va également comporter obligatoirement le **corps de la page**, ce qui sera affiché dans le navigateur. Le corps est délimité par les balises **<body> </body>**.
- A l'intérieur du corps, de très nombreuses balises sont disponibles pour mettre en forme la page :
 - les balises de niveaux de titre : **<h1> </h1>** pour le titre de niveau 1, **<h2> </h2>** pour le titre de niveau 2, etc...
 - la balise d'hyperlien : **<a> **
 - la balise d'image : ****
 - etc...

Pour aller plus loin...

- Je vous présente ici seulement quelques notions de base d'HTML qui vont permettre d'écrire un serveur HTML avec Arduino. Mais il existe de très nombreuses balises et le langage HTML est un véritable « langage » de mise en forme de page avec ses subtilités, etc...
- On trouvera sur internet toutes sortes de sites permettant d'apprendre l'HTML si on le souhaite, notamment :
 - <http://www.siteduzero.com/tutoriel-3-13666-apprenez-a-creez-votre-site-web-avec-html5-et-css3.html>
 - <http://www.w3schools.com/html/default.asp> (en anglais)
 - <http://www.w3schools.com/tags/default.asp> (toutes les balises)

Vous pouvez vous contenter de ce que je vous présente ici pour passer à la suite. Nous n'utiliserons ici que des rudiments d'HTML. Mais en même temps ça vous initie en douceur à la création de pages web. Sympa non ?

Structure de la page HTML minimale

- D'après ce que l'on vient de dire, la structure de base d'une page HTML comprend :
 - les balises de limitation du début et de fin
 - les balises de l'entête
 - les balises du corps
- A cette structure minimale, s'ajoute volontiers :
 - le titre de la page
 - la définition de l'encodage de la page

Ecrire sa première page HTML

- Ouvrir un simple éditeur de texte ou mieux un éditeur HTML (je conseille bluefish sous Ubuntu, mais il y en a pour tous les goûts)
- Copier/coller le code suivant

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8" />
    <title>Titre</title>
  </head>

  <body>
    Ma première page HTML !
  </body>

</html>
```

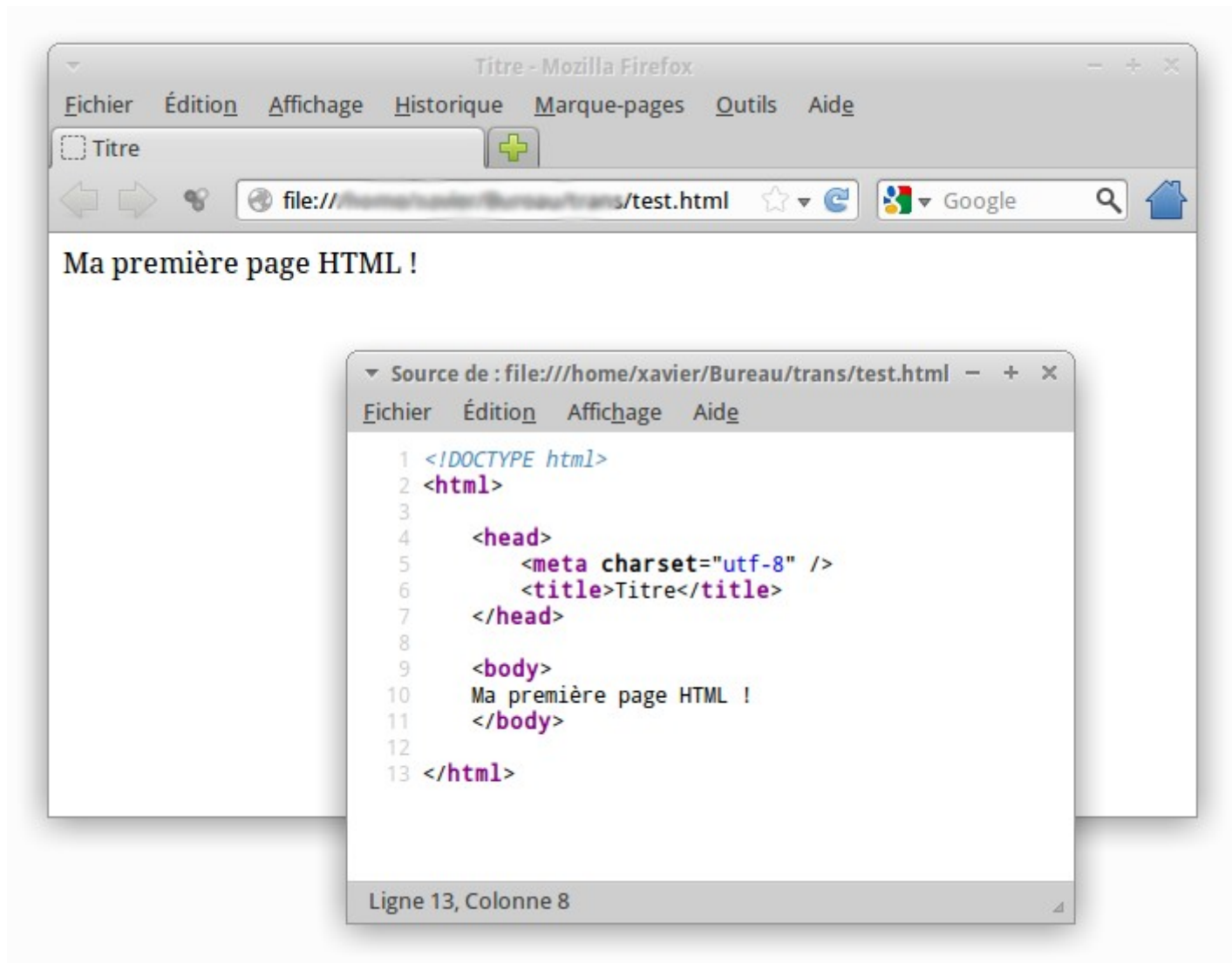
- Enregistrer le fichier au format *.html : voilà, c'est fait.

Lire une page HTML

- Pour lire une page HTML, logiquement, on utilise un navigateur, Firefox par exemple (vraiment au hasard...!)
- Puis menu Fichier > Ouvrir un fichier et sélectionner votre fichier HTML : la page doit s'afficher !

Truc : connaître le source d'une page HTML

- Quand on visualise une page HTML dans le navigateur, on voit la page, pas le code HTML, appelé aussi le source.
- Pour visualiser le source, dans Firefox, faire un clic droit dans la fenêtre de visualisation et sélectionner « Code source de la page » : vous devez voir s'afficher les secrets de la page web où vous êtes !



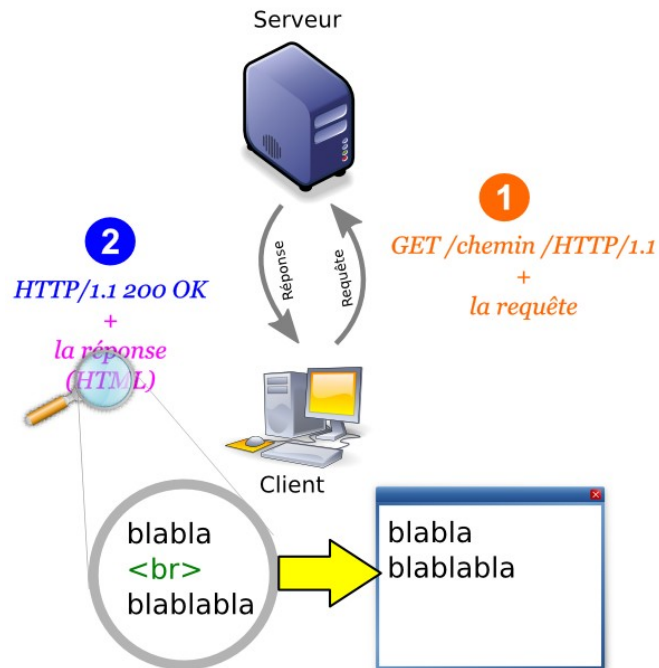
Votre première page HTML dans Firefox !

Faire un clic droit dans la fenêtre puis clic sur « Code source de la page » pour visualiser le code source de la page.

13. Notion de script « côté serveur » et de page web « dynamique »

Principe de base d'un serveur HTML

- Comme on l'a déjà dit, fondamentalement, dans son principe de base, un serveur HTML va essentiellement fournir une réponse http suivi d'une page HTML (un simple fichier texte un peu spécial...) lors de la réception d'une requête http en provenance du client.

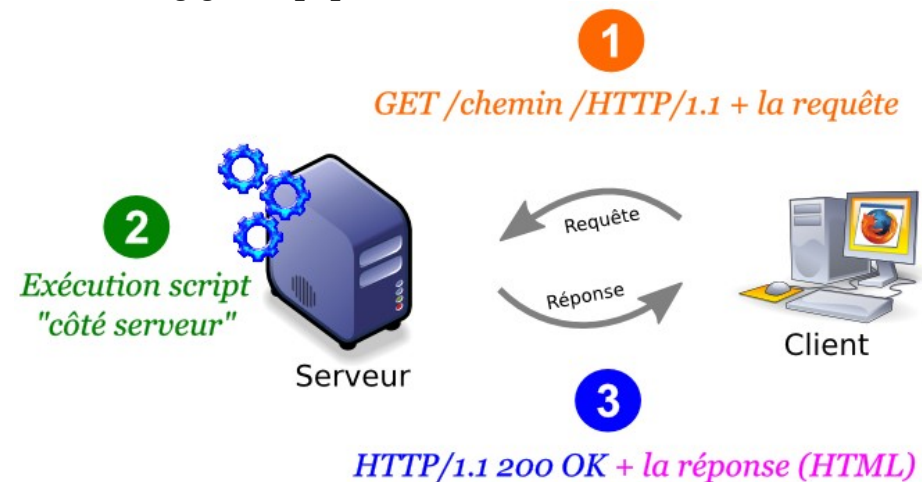


Les limites des pages « HTML » statiques

- On comprend cependant que ce principe de fonctionnement va vite présenter ses limites si l'on n'utilise que des pages HTML « statiques » c'est à dire dont le contenu est fixé d'avance.
- Par exemple, si le client envoie une réponse comportant des données, il est nécessaire que le serveur puisse les traiter et envoyer une réponse adaptée en conséquence.
- A l'inverse, il peut être intéressant qu'une page HTML une fois reçue par le client puisse avoir un fonctionnement interactif autonome sans qu'à chaque changement une nouvelle page soit envoyée par le serveur...

Script exécuté côté « serveur »

- Pour que le serveur puisse envoyer des pages HTML adaptées aux requêtes qu'il reçoit, il est nécessaire qu'un programme (ou script) puisse être exécuté « côté serveur » : ce programme va générer la page HTML à envoyer en fonction d'informations reçues, du contenu d'une base de donnée, etc...
- La page HTML ainsi produite est appelée page HTML « dynamique » car son contenu est généré par l'exécution d'un code côté serveur.
- Le plus connu de ces langages de programmation « côté serveur » est le langage libre **php**.



Remarque :

un programme Arduino « serveur » pourra exécuter du code s'apparentant à un script « côté serveur » et il n'est pas nécessaire (ni possible d'ailleurs) d'utiliser un langage tel que le php avec Arduino.

- Ici, nous ne nous étendrons pas sur le codage de script côté serveur, si ce n'est en codant notre serveur Arduino.**
- Pour info :**
 - l'utilisation du php sur un serveur présuppose que ce langage soit implémenté sur le serveur, ce qui est le cas chez la plupart des hébergeurs actuellement.
 - typiquement, le langage php est utilisé avec une base de données, notamment MySQL, d'où l'utilisation du couple Php+MySQL par de nombreuses applications web.

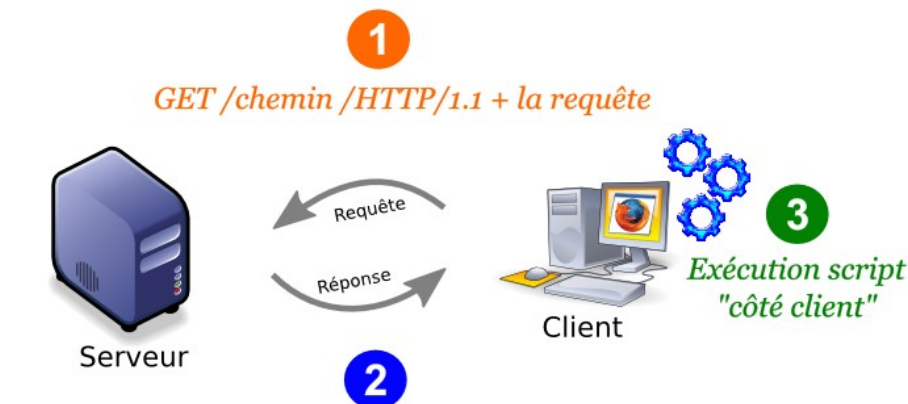
14. Notion de script « côté client » et présentation du langage Javascript

Limites d'une page web statique « côté client »

- Si le serveur envoie au client une simple page web statique, le navigateur client se contentera de l'afficher... et puis c'est tout.
- Dans ces conditions, les possibilités d'affichages dynamiques ou d'interaction avec l'utilisateur restent limitées : les formulaires HTML permettent de saisir des valeurs mais il ne sera guère possible de faire plus en HTML « pur »
- Pourtant, si on souhaite pouvoir détecter des événements utilisateurs ou si l'on souhaite réaliser un affichage dynamique dans le navigateur client, il faut pouvoir disposer d'une page capable de se modifier sans qu'un nouvel envoi soit fait par le serveur...
- La solution passe par l'utilisation d'un script « côté client »

Script exécuté « côté client »

- Ce besoin étant très réel, c'est tout naturellement qu'un langage de programmation « côté client » a été développé : c'est le javascript !
- Ainsi, dans sa réponse, le serveur va pouvoir insérer du « code » qui sera exécuté du côté du navigateur client ce qui permettra à une page web d'être interactive ou de se modifier SANS que le serveur n'envoie une nouvelle page !



GET /chemin /HTTP/1.1 + la requête

HTTP/1.1 200 OK + la réponse (HTML) + code javascript

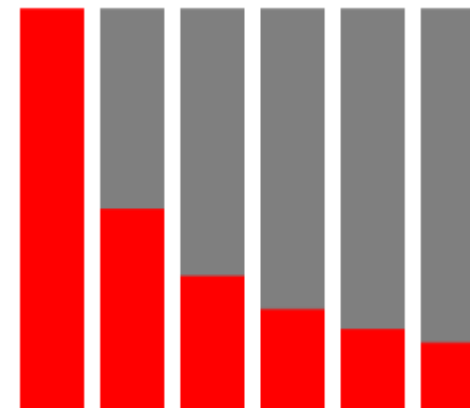
- La plupart des navigateurs intègrent le support du langage Javascript actuellement, notamment Firefox. Autrement dit, le navigateur est directement capable d'exécuter un code Javascript lorsqu'il reçoit.

Le langage Javascript

- Le langage Javascript utilise la syntaxe générale du C (et donc du langage Arduino) avec cependant quelques adaptations comparables au langage Java voire même à certains éléments du langage Python.
- En pratique, rien d'inaccessible lorsque l'on connaît Arduino !
- Pour coder, un simple éditeur texte et un navigateur suffiront pour faire ses premières armes avant d'intégrer les scripts au serveur Arduino.

Javascript et serveur Arduino

- Concrètement, un code Javascript va se présenter sous forme de texte qui sera intégré dans une page HTML ou mis dans un fichier séparé : **il va donc être très simple d'ajouter un tel script à la réponse HTML d'un serveur Arduino !**
- La question est : pourquoi faire ???
- Il faut savoir que les dernières versions d'HTML, notamment HTML 5 intègrent **un objet très intéressant : le canvas**. C'est une zone de dessin qui va pouvoir être intégrée dans une page HTML... et pour dessiner dedans ou dessus... il suffira d'écrire un petit code javascript !
- Dès lors, il va être possible, assez simplement de dessiner dans un canvas pour représenter sous forme graphique le résultat de mesures analogiques, etc...
- Bien plus, il va même être possible de faire appel à des scripts Javascript élaborés qui permettront des affichages graphiques évolués... à partir d'un simple serveur Arduino !



Exemple d'affichage graphique avec Javascript : ici les 6 voies analogiques d'Arduino

15. Syntaxe de base du langage Javascript

Intro

- Il n'est pas question, ni possible, ici, de présenter toutes les subtilités du Javascript : je vous présente simplement les bases qui vont vous permettre de démarrer à partir de ce que vous connaissez déjà du langage Arduino.

Structure

- Le Javascript utilise la même syntaxe générale que le C et donc qu'Arduino :
 - // : commentaire 1 ligne
 - /* */ : commentaire multiligne
 - ; en fin de ligne
 - { et } de limitation des sections de code des fonctions, boucles,...

Variables

- Toutes les variables, quelque soit leur type, sont déclarées avec le mot-clé **var** selon :

```
x = 0; // Une variable globale
var y = 'Hello!'; // Une autre variable globale
```

Tableaux

- Noter la possibilité de déclarer un tableau à la façon Arduino ou alors avec le mot clé **new** :

```
monTableau = [0,1,,,4,5]; // crée un tableau de longueur 6 avec 4
éléments
monTableau = new Array(0,1,2,3,4,5); // crée un tableau de longueur 6
avec 6 éléments
monTableau = new Array(365); // crée un tableau vide de longueur 365
```

Condition if

- Identique à Arduino :

```
if (expression1)
{
    //instructions réalisées si expression1 est vraie;
}
else if (expression2)
{
    //instructions réalisées si expression1 est fausse et expression2 est vraie;
}
else
{
    //instructions réalisées dans les autres cas;
}
```

Boucle For

- Idem Arduino :

```
for (var i = 0; i < 5; i++) {
    alert('Itération n°' + i);
}
```

Boucle While

- Idem Arduino :

```
while (number < 10) {
    number++;
}
```

Fonctions

- La déclaration d'une fonction se fait avec le mot clé **function** :

```
function nom_fonction(argument1, argument2, argument3) {
    instructions;

    return expression;
}
```

Déclarer un objet

- Une différence d'avec Arduino : pour déclarer un objet, on utilise le mot clé **new** selon :

```
var premierObjet = new Object();
```

Fonctions de l'objet window

- Au sein de la page web, certaines fonctions implicites attachées à l'objet window (classe DOM) sont directement accessibles :

```
alert("Hello world"); // affiche message
window.alert("Hello world"); // équivalent – affiche message
```

Pour aller plus loin

- La première chose à dire, c'est qu'à priori, **vous n'avez pas besoin d'en savoir beaucoup plus pour faire ce que je vais vous proposer ici** : vous apprendrez au fur et à mesure au besoin.
- Voici cependant quelques ressources utiles :
 - http://fr.wikipedia.org/wiki/Syntaxe_JavaScript
 - <http://www.siteduzero.com/informatique/tutoriels/dynamisez-vos-sites-web-avec-javascript>

16. Ecrire un premier script Javascript intégré dans une page HTML

De quoi avez-vous besoin ?

- De façon comparable à ce dont vous aviez besoin pour écrire une page HTML, pour écrire et exécuter vos premiers codes en script, vous allez avoir besoin :
 - d'un **éditeur HTML** à coloration syntaxique, ma préférence va à l'éditeur libre **Bluefish**, mais il y en a bien d'autres...
 - d'un **navigateur Web**, ma préférence allant à **Firefox**
- Une fois que vous avez tout ça, **vous êtes parés pour passer à l'action et écrire votre premier code Javascript.**

Pour info : différentes façon d'utiliser Javascript

- En pratique, on peut utiliser Javascript de plusieurs façon :
 - soit en insérant le code directement dans la page HTML : c'est ce que nous allons faire ici, car c'est le plus simple !
 - soit en mettant le code javascript dans un fichier séparé et en l'appelant dans la page HTML : intéressant pour des codes longs... mais nécessite un serveur pour le fichier.
 - soit en l'appelant lors d'un événement : nous ne l'utiliserons pas ici.

Balise HTML d'insertion d'un code javascript

- Logiquement, il existe une balise HTML pour insérer du code javascript au sein d'une page HTML. La balise est la suivante :

```
<script language="javascript" type="text/javascript">
<!--

    // code Javascript ici, avec sa syntaxe spécifique...

-->
</script>
```

- Dans le cas où l'on appelle un fichier externe, on fera :

```
<script src="url/fichierjavascript.js"></script>
```

Head ou Body ?

On placera typiquement le code Javascript dans le Head. Un point essentiel : une fonction javascript devra avoir été insérée AVANT d'être appelée. Le/les scripts seront exécutés ou pris en compte dans leur ordre d'apparition dans la page, de haut en bas.

Fonction onload()

- Pour éviter tout problème lié à l'insertion du code javascript au sein du code HTML, il est préférable de placer le code à exécuter au sein de la fonction onload() de l'objet window : de cette façon, on est sûr que le code Javascript sera chargé au lancement de la page web.

```
<script language="javascript" type="text/javascript">
<!--

    window.onload = function () { // au chargement de la page

        // code Javascript ici, avec sa syntaxe spécifique...

    } // fin onload

-->
</script>
```

Votre première page HTML + Javascript

- Il ne reste plus qu'à intégrer ça au sein d'une page HTML :

```
<!DOCTYPE HTML>

<!-- Debut de la page HTML -->
<html>
    <!-- Debut entete -->
    <head>
        <meta charset="utf-8" /> <!-- Encodage de la page -->
        <title>JavaScript: Test Canva </title> <!-- Titre de la page -->
        <!-- Début du code Javascript -->
        <script language="javascript" type="text/javascript">
            <!--
                window.onload = function () { // au chargement de la page
                    // code Javascript ici, avec sa syntaxe spécifique...
                    alert('hello world!');
                } // fin onload
            -->
        </script>
        <!-- Fin du code Javascript -->

    </head>
    <!-- Fin entete -->

    <!-- Debut Corps de page HTML -->
    <body >
        Ma belle page Web !

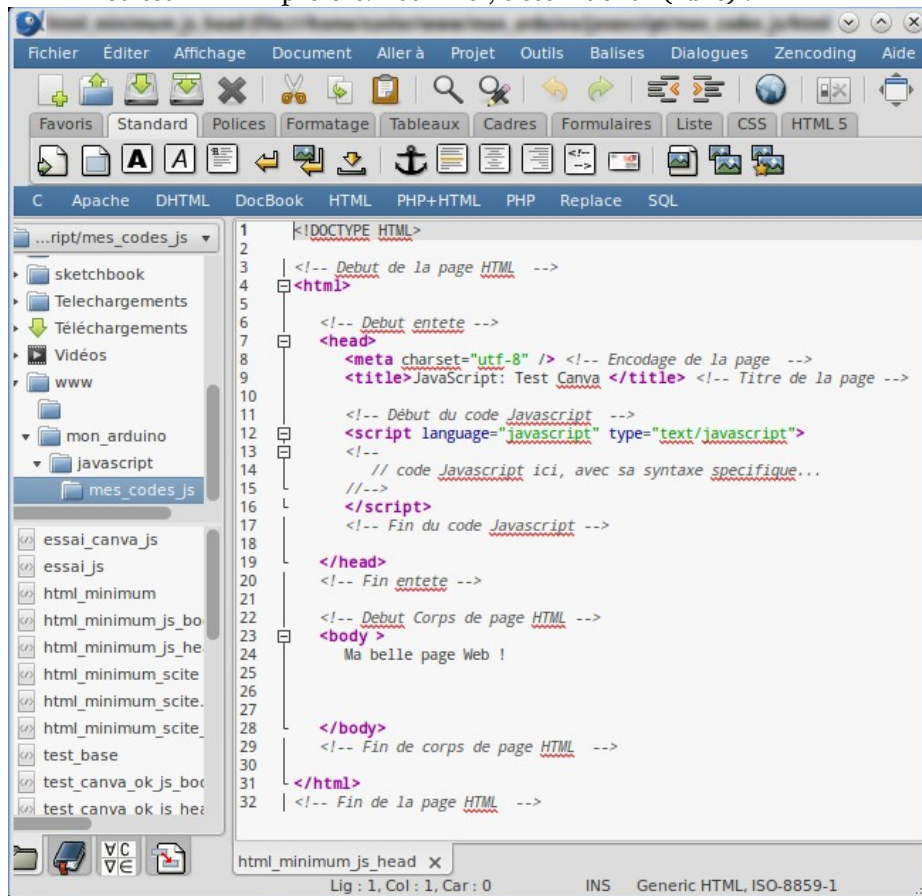
    </body>
    <!-- Fin de corps de page HTML -->

</html>
<!-- Fin de la page HTML -->
```


17. Principe de test et d'exécution d'une page HTML incluant un script javascript

Edition de la page

- Commencer par éditer le code HTML/Javascript de la page dans votre éditeur HTML préféré. Pour moi, c'est Bluefish (libre) :



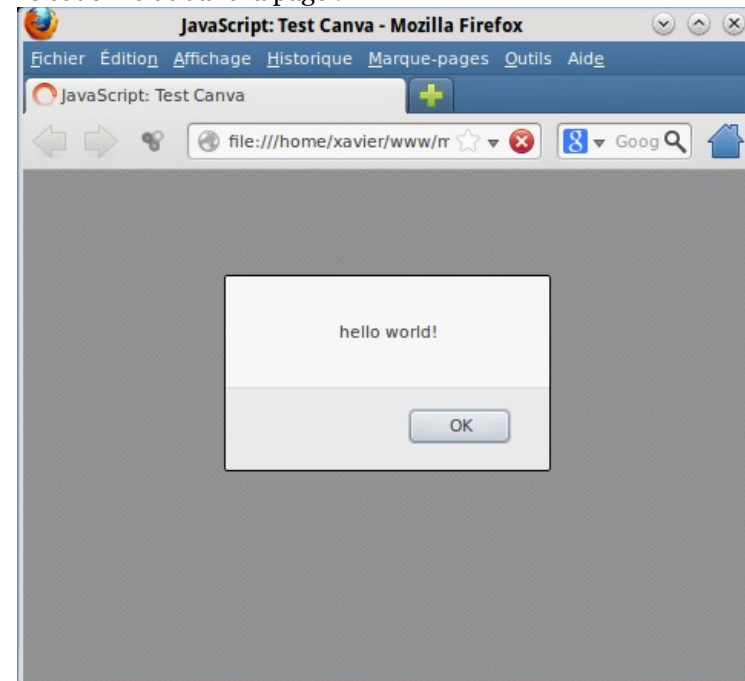
- Enregistrer le fichier au format *.html

Lancer la page avec le navigateur

- Pour lancer la page avec le navigateur :
 - soit clic sur le bouton dédié de votre éditeur (le « globe » dans bluefish)
 - soit clic droit sur le nom du fichier > ouvrir avec Firefox
 - soit ouvrir le navigateur et menu Fichier > ouvrir

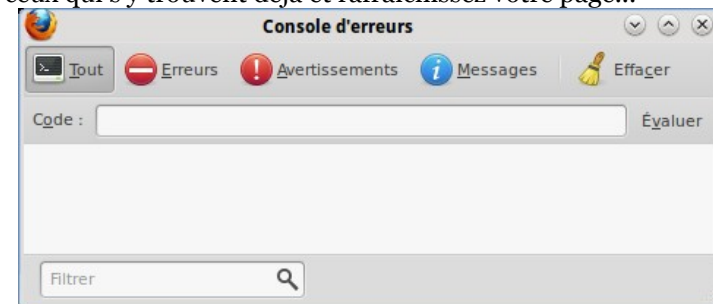
Visualiser la page dans le navigateur

- Vous devez alors voir la page s'afficher dans le navigateur en exécutant le code inclut dans la page :



Debuguer votre script

- Comme tout code, votre script peut comporter des erreurs : pour les visualiser, il faut ouvrir la « Console d'erreur » du navigateur.
- Dans Firefox : menu Outils > Developpeur web > console d'erreur
- Vous devez alors avoir une console avec les messages d'erreur : effacer ceux qui s'y trouvent déjà et rafraîchissez votre page...



18. Serveur Arduino Tcp : un premier code utilisant un code javascript

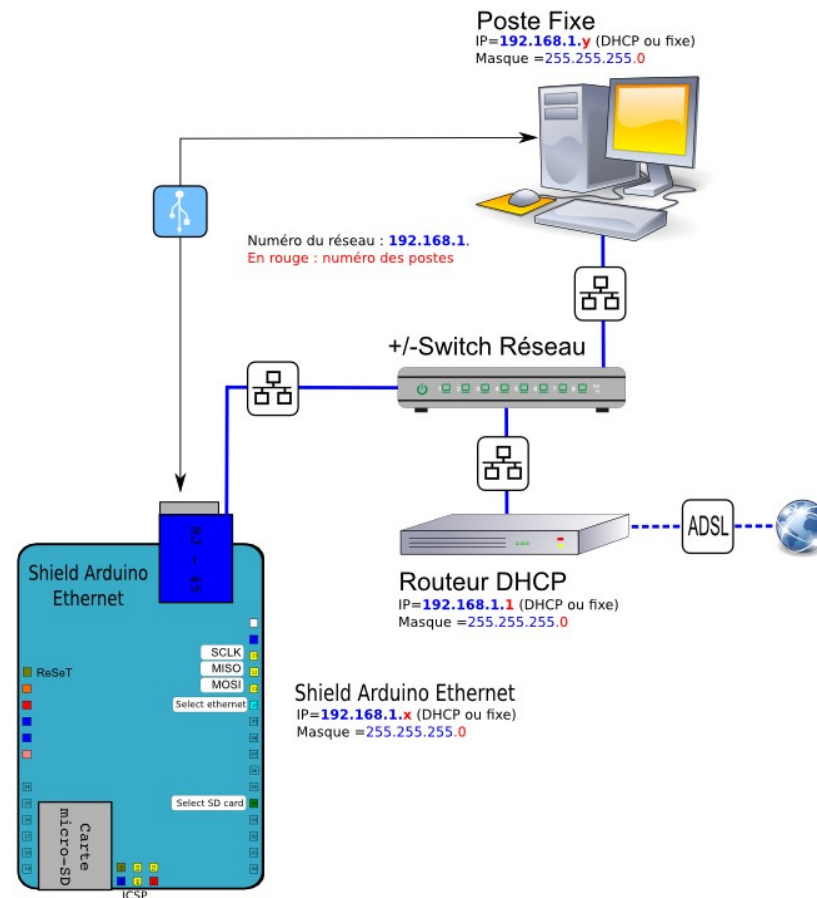
Ce qu'on va faire ici...

- Une fois que l'on a testé notre page HTML avec son bout de code Javascript, il va être possible de le tester directement à partir d'Arduino, tout simplement.

Je rappelle une nouvelle fois qu'il est nécessaire d'utiliser la version **Arduino 1.01** (ou suivante) avec les codes qui suivent.

Le schéma du réseau utilisé

- Nous reprenons ici le schéma du réseau local de base que nous avons déjà présenté par ailleurs :



Entête déclarative

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - **SPI** qui permet au shield Ethernet de communiquer avec la carte Arduino
 - et la bibliothèque **Ethernet** qui comporte toutes les fonctions nécessaires pour la communication du shield Ethernet sur le réseau Ethernet local.

Configuration du shield Ethernet

- On déclare ensuite :
 - un tableau de **byte** correspondant à l'adresse MAC du shield ethernet .
 - un ou plusieurs objets **IPAddress** correspondant aux différentes adresses IP de configuration utilisée. Ici, nous ne définirons que l'adresse IP locale du shield Ethernet.
- On déclare ensuite un objet **EthernetServer** qui configure le shield en tant que serveur. On fixe l'utilisation du port 80 (le port des connexions Web, le plus simple à utiliser car déjà ouvert par défaut sur le routeur...)

Variables utiles

- On déclare enfin des variables utiles pour la réception de la chaîne sur le réseau.

```
// --- Inclusion des bibliothèques ---

#include <SPI.h> // bibliothèque SPI - obligatoire avec bibliothèque Ethernet
#include <Ethernet.h> // bibliothèque Ethernet

// --- Déclaration des variables globales ---

//--- l'adresse mac = identifiant unique du shield
// à fixer arbitrairement ou en utilisant l'adresse imprimée sur l'étiquette du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x1A, 0x71 };

//----- l'adresse IP fixe à utiliser pour le shield Ethernet ---
IPAddress ipLocal(192,168,1,100); // l'adresse IP locale du shield Ethernet
// ATTENTION : il faut utiliser une adresse hors de la plage d'adresses du routeur DHCP
// pour connaître la plage d'adresse du routeur : s'y connecter depuis un navigateur à l'adresse xxx.xxx.xxx.1
// par exemple : sur livebox : plage adresses DHCP entre .10 et .50 => on peut utiliser .100 pour le shield ethernet

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

//--- création de l'objet serveur ---
EthernetServer serveurHTTP(80); // crée un objet serveur utilisant le port 80 = port HTTP

String chaîneRecue=""; // déclare un string vide global pour réception chaîne requête
int comptChar=0; // variable de comptage des caractères reçus
```

Fonction **setup()**

Initialisation série

- On initialise la connexion série

Initialisation du shield Ethernet

- On initialise le module Ethernet avec la fonction `Ethernet.begin()`. Bien comprendre que cette fonction initialise simplement le shield Ethernet d'un point de vue matériel. A ce stade, il n'est configuré ni en serveur, ni en client.

Affichage de l'adresse IP du shield Ethernet

- On affiche l'adresse IP attribuée au module. Remarquer que l'instruction `print` supporte l'objet `IPAddress`.

Initialisation du serveur

- Logiquement, on initialise le serveur à l'aide de l'instruction `begin()`

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter 1 seule fois au démarrage du programme ---

// ----- Initialisation fonctionnalités utilisées -----

Serial.begin(115200); // Initialise connexion Série

//---- initialise la connexion Ethernet avec l'adresse MAC du module Ethernet, l'adresse IP Locale
//---- +/- l'adresse IP du serveurDNS , l'adresse IP de la passerelle internet et le masque du réseau local

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - utilise plus de mémoire Flash (env + 6Ko)
Ethernet.begin(mac, ipLocal); // forme conseillée pour fixer IP fixe locale
//Ethernet.begin(mac, ipLocal, serverDNS, passerelle, masque); // forme complète

delay(1000); // donne le temps à la carte Ethernet de s'initialiser

Serial.print("Shield Ethernet OK : L'adresse IP du shield Ethernet est : " );

Serial.println(Ethernet.localIP());

//---- initialise le serveur ----
serveurHTTP.begin();
Serial.println("Serveur Ethernet OK : Ecoute sur port 80 (http)");

} // fin de la fonction setup()
```

Fonction **loop()** (1)

Déclaration d'un objet client

- On commence par créer un objet **EthernetClient** qui sera local à la boucle **loop()** : ce client existera seulement si une connexion entrante existe, ce qui est testé à l'aide de la fonction **.available()** de l'objet **EthernetServer** précédemment configuré.

Réception des caractères

- Ensuite, si le client existe, après avoir affiché quelques messages,...
- on teste si le client est connecté : ceci est testé à l'aide de la fonction **.connected()** de l'objet **EthernetClient**.
- Puis, à l'aide d'une boucle **while()** et de la fonction **.available()** de l'objet **EthernetClient**, qui bouclera tant qu'un caractère sera présent : on affiche le caractère reçu et on l'ajoute à une chaîne de réception
- Une condition permet d'éviter la surcharge en réception au delà de 100 caractères.

```
void loop(){ // debut de la fonction loop()

// crée un objet client basé sur le client connecté au serveur
EthernetClient client = serveurHTTP.available();

if (client) { // si l'objet client n'est pas vide
// le test est VRAI si le client existe

// message d'accueil dans le Terminal Série
Serial.println ("-----");
Serial.println ("Client present !");
Serial.println ("Voici la requete du client:");

////////// Réception de la chaine de la requete //////////

//-- initialisation des variables utilisées pour l'échange serveur/client
chaineRecue=""; // vide le String de reception
comptChar=0; // compteur de caractères en réception à 0

if (client.connected()) { // si le client est connecté

////////// Réception de la chaine par le réseau //////////
while (client.available()) { // tant que des octets sont disponibles en lecture
// le test est vrai si il y a au moins 1 octet disponible

char c = client.read(); // l'octet suivant reçu du client est mis dans la variable c
comptChar=comptChar+1; // incrémente le compteur de caractère reçus

Serial.print(c); // affiche le caractère reçu dans le Terminal Série

//--- on ne mémorise que les n premiers caractères de la requete reçue
//--- afin de ne pas surcharger la RAM et car cela suffit pour l'analyse de la requete
if (comptChar<=100) chaineRecue=chaineRecue+c; // ajoute le caractère reçu au String pour les N premiers caractères
//else break; // une fois le nombre de caractères dépassés sort du while

} // --- fin while client.available = fin "tant que octet en lecture"

Serial.println ("Reception requete terminee");
```


Fonction **loop()** (2) : Affichage et analyse de la chaîne reçue

- Ensuite, tout simplement, on affiche la chaîne reçue
- Puis on teste si la chaîne commence bien l'entête GET attendue dans le cas de la réception d'une requête HTTP valide :

```
//////////////////// Affichage de la requete reçue //////////////////////
Serial.println(F("----- Affichage de la requete recue -----")); // affiche le String de la requete
Serial.println (F("Chaîne prise en compte pour analyse : "));
Serial.println(chaineRecue); // affiche le String de la requete pris en compte pour analyse

//////////////////// Analyse de la requete reçue //////////////////////
Serial.println(F("----- Analyse de la requete recue -----")); // analyse le String de la requete

//----- analyse si la chaîne reçue est une requete GET -----
if (chaineRecue.startsWith("GET")) { // si la chaîne recue commence par GET

    Serial.println (F("Requete HTTP valide !"));

//----- +/- extraction et analyse de la sous-chaîne utile -----
```

Fonction **loop()** (3) : Envoi de la réponse Http

- on envoie ensuite une réponse HTTP valide, affichée également en copie dans le terminal série :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
////////// Réponse HTTP suivie de la Page HTML de réponse //////////  
  
//-- envoi de la réponse HTTP ---  
client.println("HTTP/1.1 200 OK"); // entete de la réponse : protocole HTTP 1.1 et exécution requete réussie  
client.println("Content-Type: text/html"); // précise le type de contenu de la réponse qui suit  
client.println("Connection: close"); // précise que la connexion se ferme après la réponse  
client.println(); // ligne blanche  
  
//--- envoi en copie de la réponse http sur le port série  
Serial.println("La reponse HTTP suivante est envoyee au client distant :");  
Serial.println("HTTP/1.1 200 OK");  
Serial.println("Content-Type: text/html");  
Serial.println("Connection: close");  
Serial.println();  
  
//--- la réponse HTML à afficher dans le navigateur
```

Fonction **loop()** (4) : Envoi du début et du head de la page HTML de réponse

- Par un jeu de **println()**, on envoie la page HTML avec :
 - les balises **<html>** et **</html>** de début et fin de page
 - les balises **<head>** et **</head>** d'entête
 - **<body>** et **</body>** du corps de la page
- **A ce niveau, nous insérons la balise script et nous insérons à l'aide println successifs le code javascript vu précédemment .** Remarquer l'utilisation de la fonction **window.onload()** : il sera ainsi lancé au chargement de la page et ce qui permet de le placer dans le head bien que le script puisse potentiellement utiliser des éléments placés dans le body.
- Remarquer également l'utilisation de la balise **<center>** qui permet de centrer le contenu de la page HTML

```
//----- début de la page HTML -----
client.println("<!DOCTYPE html>");
client.println("<html>");

//----- head = entete de la page HTML -----
client.println("<head>");

client.println("<meta charset=\"utf-8\" />"); // fixe encodage caractères - utiliser idem dans navigateur
client.println("<title>Titre</title>");// titre de la page HTML

//===== bloc de code javascript =====
client.println("<!-- Début du code Javascript -->");
client.println("<script language=\"javascript\" type=\"text/javascript\">");
client.println("<!--      ");

client.println("window.onload = function () { // au chargement de la page");

    client.println("// code Javascript ici, avec sa syntaxe specifique...");
    client.println("alert('hello world!');");

client.println("} // fin onload");

client.println("//-->");
client.println("</script>");
client.println("<!-- Fin du code Javascript --> ");
//===== fin du bloc de code javascript =====

client.println("</head>");
//----- fin head = fin entete de la page HTML -----
```

Fonction **loop()** (5) : envoi du body et de la fin de la page HTML de réponse

- De la même façon, par un jeu de **println()**, on envoie la suite du code HTML au besoin.
- Suivent les balises de clôture de la page web

```
//----- body = corps de la page HTML -----
client.println("<body>");

// affiche chaines caractères simples
client.println("<CENTER>"); //pour centrer la suite de la page
client.println("<br>");
client.println("Serveur Arduino : Test de page HTML incluant du code Javascript");
client.println("<br>");

// intègre une image - suppose connexion internet disponible
//client.println("<CENTER> <img src=\"http://www.arduino.cc/mes_images/communs/led_rouge_5mm.gif\"> </CENTER>");
//client.println("<br>");

    client.println("</body>");
//----- fin body = fin corps de la page -----

client.println("</html>");
//----- fin de la page HTML -----

} // fin if GET
```

Fonction **loop()** (6) : Fermeture de la connexion avec le client

- si la requête reçue n'est pas une « GET », un message indique que la requête n'est pas valide.
- Puis la connexion avec le client est clotûrée.

```
else { // si la chaine recue ne commence pas par GET
  Serial.println (F("Requete HTTP non valide !"));
} // fin else

//----- fermeture de la connexion -----

// fermeture de la connexion avec le client après envoi réponse
delay(1); // laisse le temps au client de recevoir la réponse
client.stop();
Serial.println(F("----- Fermeture de la connexion avec le client -----")); // affiche le String de la requete
Serial.println (F(""));

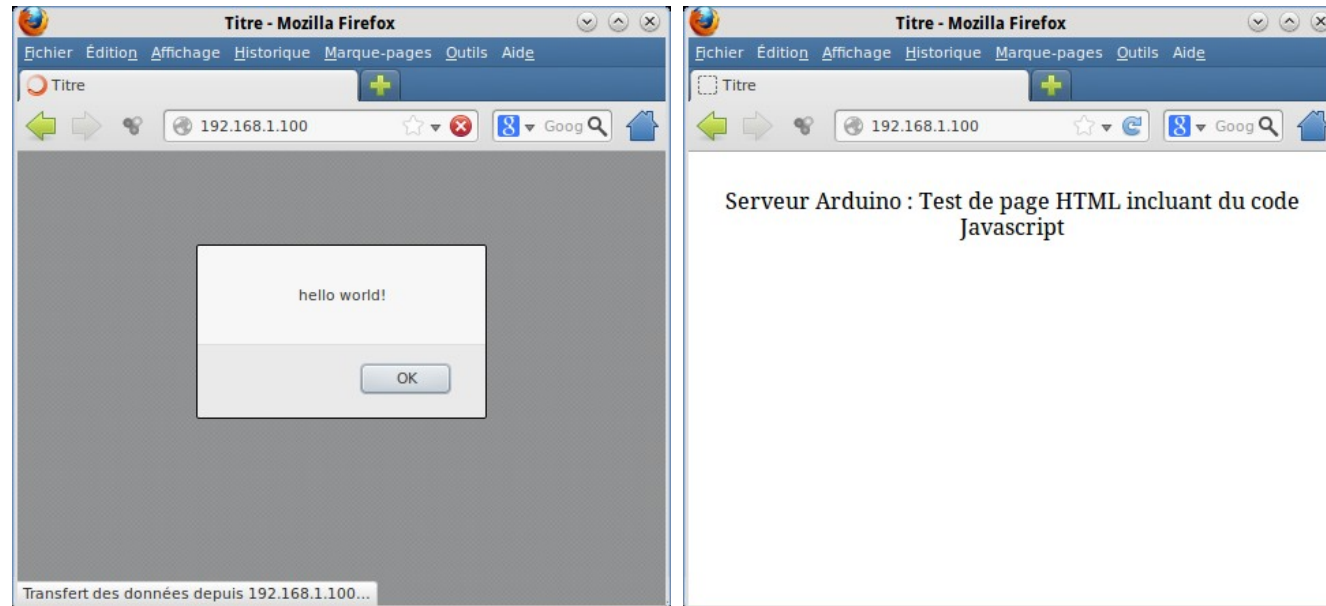
} // --- fin if client connected

} //---- fin if client ----

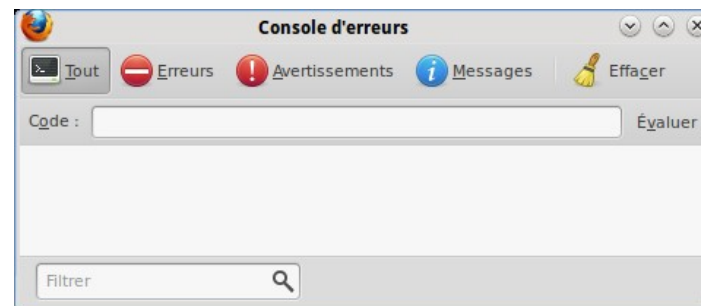
} // fin de la fonction loop()
```


Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80)
- A présent, **ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse** (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Dans le navigateur, on doit ici voir un popup avec « Hello World ! » affiché. Cliqué dessus : le texte de la page s'affiche.



Un simple popup s'affiche au lancement de la page web :
tout bête, mais au moins vous êtes sûr que le premier code Javascript de votre serveur Arduino fonctionne !

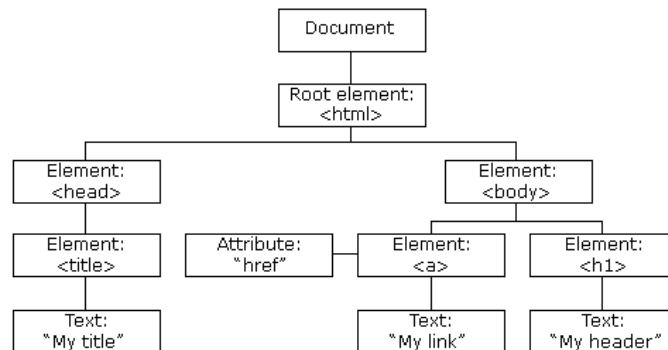


La console d'erreur est vide : bravo !

19. Pour info : le DOM, l'accès aux éléments d'une page HTML et les fonctions de l'objet window

Le DOM

- Le DOM, ou **Document Model Object**, est un standard du web qui permet de décrire et d'accéder aux différents éléments d'une interface, et notamment d'une page web, à partir de tout langage de programmation et notamment le Javascript.
- Chaque élément d'une page web va ainsi pouvoir être facilement désigné et être modifié/utilisé au sein du code Javascript.
- D'un point de vue conceptuel, la page HTML ou le document est vu au sein du DOM sous la forme d'une arborescence dont chaque élément est un nœud :



- Les éléments qui dépendent d'un nœud sont appelés les **enfants** (child) de ce nœud, et le nœud dont dépendent d'autres éléments est appelé **parent**.
- Bien plus, le DOM va permettre de **détecter la survenue d'événement au sein de la page web** : le moment où elle est chargée (onload), où un click souris survient (onclick), où elle est fermée, etc...
- Pour plus de détails, voir :
 - http://fr.wikipedia.org/wiki/Document_Object_Model
 - http://www.w3schools.com/html/dom/dom_intro.asp

Accéder à un élément

- Pour accéder à un élément de la page, on utilisera la fonction `getElementById(« nom »)` :

```
var element=document.getElementById("intro");
```

Associer un élément HTML à un événement du DOM

- La plupart des éléments HTML peuvent être associés à un événement sous la forme (où nomFonction est la fonction Javascript à appeler) :

```
<element onload="nomFonction(param)">
```

Accéder au contenu d'une balise HTML

- Pour accéder au contenu inséré dans une balise HTML, on utilisera la propriété `innerHTML` d'un objet donné :

```
<p id="intro">Hello World!</p>
```

```
<script>
var txt=document.getElementById("intro").innerHTML;
document.write(txt);
</script>
```

Les fonctions disponibles

- Les fonctions disponibles pour gérer, modifier, créer, ajouter, supprimer des éléments du DOM sont très nombreuses.
- Nombreuses également les fonctions permettant de capturer les événements.
- Il n'est pas possible ici d'en dire davantage : nous présenterons les fonctions utilisées au besoin.
- Pour en apprendre plus :

http://www.w3schools.com/html/dom/dom_intro.asp

A part, l'objet implicite window

- Quand une page web est affichée dans un navigateur, un objet implicite est créé, attaché au navigateur, et appelé window
- L'objet window représente la fenêtre du navigateur dans laquelle la page web est affichée.
- Les fonctions de l'objet window ont la particularité d'être accessibles directement. Ces fonctions sont nombreuses et utilisées fréquemment au sein d'un code javascript.
- L'une d'entre elle est la fonction `alert()` qui ouvre un popup avec un bouton OK. Cette fonction est donc accessible de 2 façons :

```
alert("Hello world"); // affiche message
window.alert("Hello world"); // équivalent – affiche message
```

- Un événement de l'objet window utile est notamment `onload` qui permet d'attendre que tous les éléments de la page soient chargés avant d'utiliser le code javascript (voir http://www.w3schools.com/tags/ref_eventattributes.asp) :

```
window.onload = function () {
```

Pour aller plus loin :

- <http://www.w3schools.com/jsref/default.asp>
- Les événements HTML5 : http://www.w3schools.com/tags/ref_eventattributes.asp

20. HTML : Présentation de l'objet canvas

Présentation

- Il existe de très nombreux objets HTML et il n'est pas question de les passer en revue ici, mais il y en a un qui mérite toute notre attention : le canvas !
- Un canvas est un objet HTML5 particulièrement intéressant : il s'agit d'un objet qui représente une zone de dessin 2D que l'on va pouvoir intégrer au sein d'une page HTML.
- Le très grand intérêt du canvas, c'est qu'il dispose de nombreuses fonctions de dessin qui vont permettre d'y dessiner simplement ce que l'on veut, et donc, dans notre cas, de présenter des données en provenance d'Arduino sous forme graphique dans une page web, ni plus ni moins !

Balise d'insertion

- La balise HTML permettant d'intégrer un canvas dans une page est tout ce qu'il y a de plus classique :
 - une balise de début `<canvas>` et de fin `</canvas>`
 - des paramètres définissant le canvas, notamment :
 - un identifiant
 - une largeur et une hauteur en pixels
- Ce qui nous donne :

```
<canvas id="cvs" width="300" height="300"></canvas>
```

- Ici, on crée un canvas, autrement dit une zone de dessin :
 - appelée cvs
 - de 300 pixels de large sur 200 pixels de haut
- Difficile de faire plus simple...

Page HTML utilisant un canvas

```
<!DOCTYPE HTML>

<!-- Debut de la page HTML -->
<html>

  <!-- Debut entete -->
  <head>
    <meta charset="utf-8" /> <!-- Encodage de la page -->
    <title>Test Canva seul</title> <!-- Titre de la page -->
  </head>
  <!-- Fin entete -->

  <!-- Debut Corps de page HTML -->
  <body>
    <canvas id="papier" width="300" height="300"></canvas>
    <br />
    Exemple de Canva
  </body>
  <!-- Fin de corps de page HTML -->

</html>
<!-- Fin de la page HTML -->
```

Page HTML + Javascript utilisant un canvas

- L'utilisation la plus utile d'un canvas est de le coupler à du code Javascript qui va permettre de dessiner à l'intérieur, ce qui donne :

```
<!DOCTYPE HTML>

<!-- Debut de la page HTML -->
<html>

  <!-- Debut entete -->
  <head>

    <meta charset="utf-8" /> <!-- Encodage de la page -->
    <title>JavaScript: Test Canva </title> <!-- Titre de la page -->

    <!-- Debut du code Javascript -->
    <script language="javascript" type="text/javascript">
      <!--
      window.onload = function() {

        var canvas = document.getElementById("cvs"); // declare
        objet canvas a partir nom

        // mettre ici le code de dessin dans le canvas

      } // fin window.onload
      <!-->
    </script>
    <!-- Fin du code Javascript -->

  </head>
  <!-- Fin entete -->

  <!-- Debut Corps de page HTML -->
  <body >

    <canvas id="cvs" width="300" height="300"></canvas>
    <!-- IMPORTANT : le canvas doit etre declare AVANT le code qui
    l'utilise ! -->

    <br />
    Exemple de Canva

  </body>
  <!-- Fin de corps de page HTML -->

</html>
<!-- Fin de la page HTML -->
```

En savoir plus

- Toutes les méthodes et propriétés de l'objet canvas :
http://www.w3schools.com/tags/ref_canvas.asp
- Une petite page qui permet facilement de tester les possibilités du canvas :
<http://jm.davalan.org/lang/jsc/js09.html>

21. Javascript : Les fonctions essentielles de l'objet canvas

Déclaration

- La première chose à faire au niveau du code Javascript qui va utiliser le Canvas, c'est de créer l'objet représentant le canvas, ce qui se fait avec la fonction `getElementById()` vue précédemment :

```
var canvas = document.getElementById("nomCanvas"); // declare objet canvas a partir id = nom
```

Initialisation

- Une fois l'objet Canvas déclaré, on doit avant toute chose l'initialiser à l'aide d'une fonction particulière appelée `getContext()` et qui renvoie un objet Context qui servira pour appeler les fonctions de dessin (le canvas en lui-même n'est qu'un conteneur).
- Cette fonction reçoit en paramètre « 2d » pour signifier le type de dessin qui va être effectué.

La 3D dans un Canvas est possible et arrive progressivement.
Voir ici par exemple : <http://www.canvasdemos.com/type/applications/3d-applications/>

- On a :

```
var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin
```

- En pratique cependant, on teste au préalable le retour de la fonction `getContext()` avant d'appeler les fonctions de dessin, ce qui donne :

```
if (canvas.getContext){ // la fonction getContext() renvoie True si canva accessible

    var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin

    // fonctions de dessin ici
}
else {

    // code si canvas non disponible
}
```

Les fonctions de dessin de base

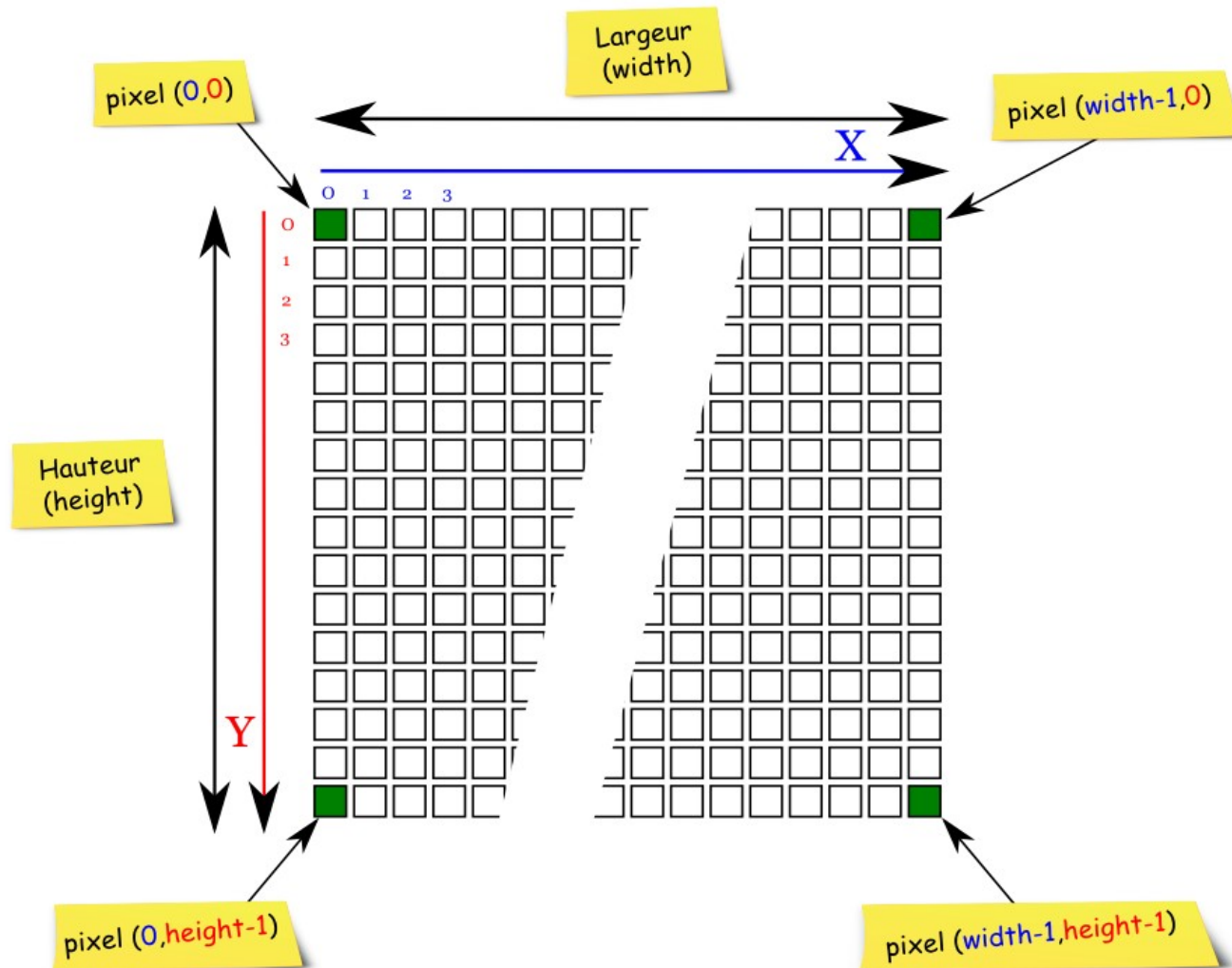
- Une fois que l'on dispose de l'objet Context d'accès aux fonctions de dessin du Canvas, on va pouvoir passer à l'action : **en fait, à ce stade, c'est tout un nouveau monde de possibilités qui s'ouvre à vous !** Un peu à la façon Processing pour ceux qui connaissent...
- Tout d'abord, on peut modifier les dimensions du Canvas avec les propriétés `.width` et `.height`
- On peut également paramétrer le mode de dessin avec :
 - `fillStyle()` : pour fixer la couleur de remplissage
 - `strokeStyle()` : pour fixer la couleur de pourtour
 - etc...
- On dispose bien sûr des fonctions géométriques de base :
 - `strokeRect()` : pourtour d'un rectangle
 - `fillRect()` : un rectangle plein
 - etc...
- On dispose également d'une possibilité de tracer un élément sous la forme d'un « chemin » de points successifs avec les fonctions :
 - `beginPath()` et `closePath()`
 - `lineTo()`
 - `moveTo()`
 - `arc()` et `arcTo()`
 - etc...
- Pour plus de détails, voir : http://www.w3schools.com/tags/ref_canvas.asp
- Voici un exemple simple traçant un carré vert :

```
var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin
```

```
// le code graphique ci-dessous
ctx.fillStyle = "rgb(0,500,0)"; // couleur remplissage
ctx.fillRect (50, 50, 200, 200); // rectangle plein
```

22. Objet Canvas : système de coordonnées

- Le système de coordonnées d'un canvas de largeur width et de hauteur height est le suivant :



23. Exemple de page HTML+ Javascript : dessiner dans un canvas

Ce que l'on va faire ici

- Cette fois, vous allez écrire votre premier code Javascript utilisant un canva : prêt, c'est parti ! Pour le moment, vous allez le faire directement sur votre ordinateur, ensuite nous le ferons à partir d'Arduino !

Le code Javascript

Structure générale

- On commence par encadrer le code au sein d'une fonction appelée lors de la survenue de l'évènement **onload** de l'objet window (qui représente la fenêtre du navigateur où est chargée la page) pour que le code javascript ne soit appelé que lorsque tous les éléments de la pages HTML ont été chargés.
- On déclare ensuite l'objet canvas que l'on récupère (à partir de son nom utilisé dans le code HTML) à l'aide de la fonction **getElementById()**
- On peut au besoin reparamétrer la taille du canvas, mais ça n'est pas obligatoire
- Ensuite, on teste l'existence du canvas à l'aide de la propriété **getContext** qui renvoie True si le canvas existe : on place le code graphique au sein de cette condition.

Code graphique

- On commence le code graphique à proprement parler (placé au sein de la condition vérifiant l'existence du canvas) en créant l'objet Context qui va donner accès aux fonctions graphiques du canvas : ceci se fait à l'aide de la fonction **getContext(« 2d »)** de l'objet canvas,
- Ensuite, on commence par tracer un rectangle plein de la taille du canvas en fixant la couleur rgb au préalable
- Puis on trace un carré vert.

```
window.onload = function() {  
  
    var canvas = document.getElementById("nomCanvas"); // declare objet canvas a partir id = nom  
  
    canvas.width = 300; // largeur canvas  
    canvas.height = 300; // hauteur canvas  
  
    if (canvas.getContext){ // la fonction getContext() renvoie True si canvas accessible  
  
        var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin  
  
        // le code graphique ci-dessous  
        // carre gris de la taille du canvas  
        ctx.fillStyle = "rgb(200,200,200)"; // couleur de remplissage rgb 0-255  
        ctx.fillRect (0, 0, canvas.width, canvas.height); // rectangle  
        // carre vert  
        ctx.fillStyle = "rgb(0,255,0)"; // couleur remplissage  
        ctx.fillRect (50, 50, 200, 200); // rectangle  
  
    } // fin si canvas existe  
  
    else {  
  
        // code si canvas non disponible  
  
    }  
  
} // fin window.onload
```

Le code HTML + Javascript complet (1) : le head

- On intègre ce code au niveau du Head de la page HTML

```
<!-- Debut de la page -->
<html>

  <!-- Debut entete -->
  <head>

    <meta charset="utf-8" /> <!-- Encodage de la page -->
    <title>JavaScript: Test Canva </title> <!-- Titre de la page -->

    <!-- Debut du code Javascript -->
    <script language="javascript" type="text/javascript">
    <!--
window.onload = function() {

    var canvas = document.getElementById("nomCanvas"); // declare objet canvas a partir id = nom

    canvas.width = 300; // largeur canvas
    canvas.height = 300; // hauteur canvas

    if (canvas.getContext()){ // la fonction getContext() renvoie True si canvas accessible

        var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin

        // le code graphique ci-dessous

        // carre gris de la taille du canvas
        ctx.fillStyle = "rgb(200,200,200)"; // couleur de remplissage rgb 0-255
        ctx.fillRect (0, 0, canvas.width, canvas.height); // rectangle

        // carre vert
        ctx.fillStyle = "rgb(0,255,0)"; // couleur remplissage
        ctx.fillRect (50, 50, 200, 200); // rectangle

    } // fin si canvas existe

    else {
        // code si canvas non disponible
    } // fin else

} // fin window.onload
    <!-->
    </script>
    <!-- Fin du code Javascript -->

  </head>
  <!-- Fin entete -->
```

Le code HTML + Javascript complet (2) : le body

- Au niveau du body, les choses sont simples : on insère simplement une balise canvas avec les paramètres définissant le canvas, notamment :
 - le nom (le même que celui utilisé avec la fonction `getElementById()` dans le code javascript `+++`)
 - une largeur et une hauteur en pixels
- le reste de la page HTML, ici limitée à un simple message texte,
- ce qui nous donne :

```
<!-- Debut Corps de page  -->
<body >

    <canvas id="nomCanvas" width="300" height="300"></canvas>

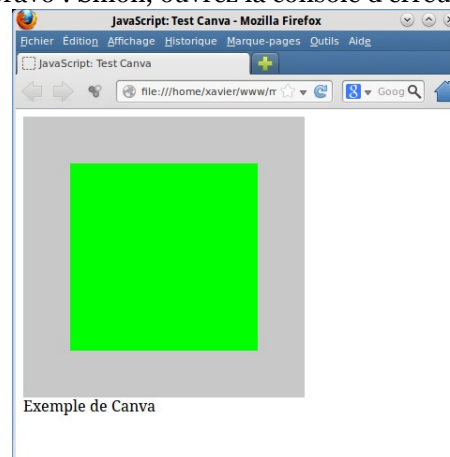
    <br />
    Exemple de Canva

</body>
<!-- Fin de corps de page  -->

</html>
<!-- Fin de la page  -->
```

Résultat

- Ensuite, il suffit de lancer votre page dans le navigateur (je conseille Firefox) :
 - soit clic sur le bouton dédié de votre éditeur (le « globe » dans bluefish)
 - soit clic droit sur le nom du fichier > ouvrir avec Firefox
 - soit ouvrir le navigateur et menu Fichier > ouvrir
- Vous devez obtenir l'affichage suivant : si c'est le cas, bravo ! Sinon, ouvrez la console d'erreur du navigateur et vérifiez ce qui ne vas pas.



24. Serveur Arduino : Envoyer une page HTML + Javascript et afficher un canvas dans le navigateur client

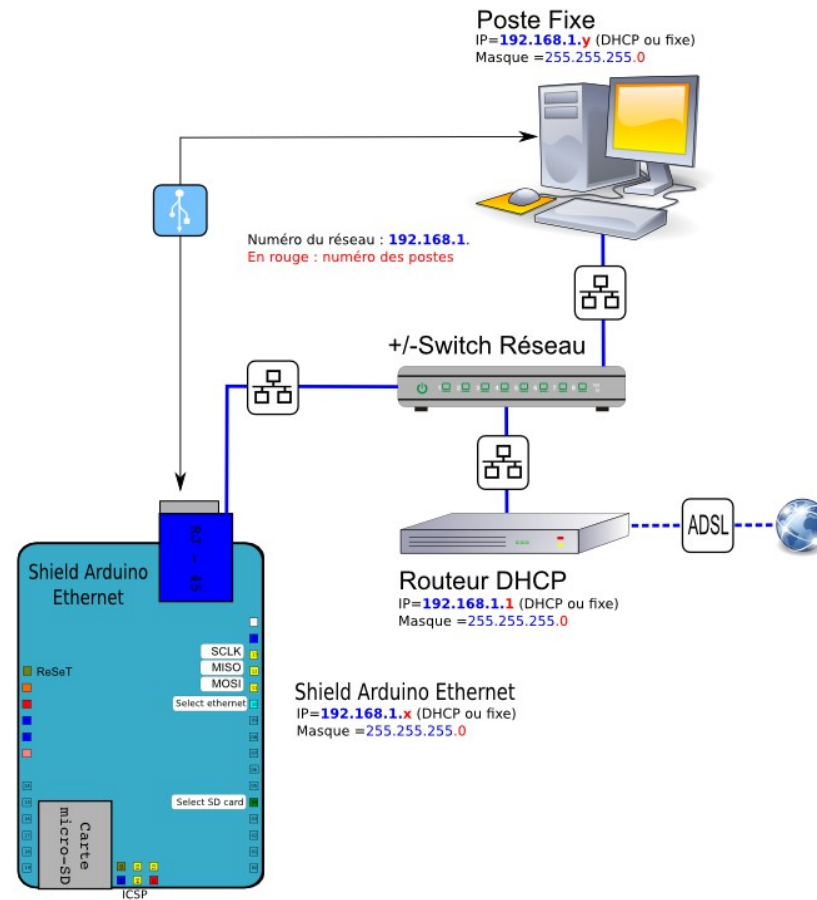
Ce qu'on va faire ici...

- Vous êtes toujours là ? Courage, le plus dur est fait ! A présent que vous avez pu tester votre page HTML+Javascript sur votre PC, il ne vous reste plus qu'à l'intégrer à notre code Arduino... Rien de bien sorcier. Allez, on continue !
- Remarquer au passage comment vous pouvez dissocier le développement d'un serveur Arduino « élaboré » en 2 temps :
 - tout d'abord test et mise au point en local, sur votre ordinateur, de la page HTML + Javascript, tranquillement
 - ensuite seulement, intégration de la page dans votre code Arduino
 - au final, cette procédure « progressive » et modulaire vous permet de maîtriser chaque aspect de l'application finale

Je rappelle une nouvelle fois qu'il est nécessaire d'utiliser la version **Arduino 1.01** (ou suivante) avec les codes qui suivent.

Le schéma du réseau utilisé

- Nous reprenons ici le schéma du réseau local de base que nous avons déjà présenté par ailleurs :



Entête déclarative

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - **SPI** qui permet au shield Ethernet de communiquer avec la carte Arduino
 - et la bibliothèque **Ethernet** qui comporte toutes les fonctions nécessaires pour la communication du shield Ethernet sur le réseau Ethernet local.

Configuration du shield Ethernet

- On déclare ensuite :
 - un tableau de **byte** correspondant à l'adresse MAC du shield ethernet .
 - un ou plusieurs objets **IPAddress** correspondant aux différentes adresses IP de configuration utilisée. Ici, nous ne définirons que l'adresse IP locale du shield Ethernet.
- On déclare ensuite un objet **EthernetServer** qui configure le shield en tant que serveur. On fixe l'utilisation du port 80 (le port des connexions Web, le plus simple à utiliser car déjà ouvert par défaut sur le routeur...)

Variables utiles

- On déclare enfin des variables utiles pour la réception de la chaîne sur le réseau.

```
// --- Inclusion des bibliothèques ---  
  
#include <SPI.h> // bibliothèque SPI - obligatoire avec bibliothèque Ethernet  
#include <Ethernet.h> // bibliothèque Ethernet  
  
// --- Déclaration des variables globales ---  
  
//--- l'adresse mac = identifiant unique du shield  
// à fixer arbitrairement ou en utilisant l'adresse imprimée sur l'étiquette du shield  
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x1A, 0x71 };  
  
//----- l'adresse IP fixe à utiliser pour le shield Ethernet ---  
IPAddress ipLocal(192,168,1,100); // l'adresse IP locale du shield Ethernet  
// ATTENTION : il faut utiliser une adresse hors de la plage d'adresses du routeur DHCP  
// pour connaître la plage d'adresse du routeur : s'y connecter depuis un navigateur à l'adresse xxx.xxx.xxx.1  
// par exemple : sur livebox : plage adresses DHCP entre .10 et .50 => on peut utiliser .100 pour le shield ethernet  
  
// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---  
  
//--- création de l'objet serveur ---  
EthernetServer serveurHTTP(80); // crée un objet serveur utilisant le port 80 = port HTTP  
  
String chaineRecue=""; // déclare un string vide global pour réception chaîne requête  
int comptChar=0; // variable de comptage des caractères reçus
```


Fonction **setup()**

Initialisation série

- On initialise la connexion série

Initialisation du shield Ethernet

- On initialise le module Ethernet avec la fonction `Ethernet.begin()`. Bien comprendre que cette fonction initialise simplement le shield Ethernet d'un point de vue matériel. A ce stade, il n'est configuré ni en serveur, ni en client.

Affichage de l'adresse IP du shield Ethernet

- On affiche l'adresse IP attribuée au module. Reremarquer que l'instruction `print` supporte l'objet `IPAddress`.

Initialisation du serveur

- Logiquement, on initialise le serveur à l'aide de l'instruction `begin()`

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter 1 seule fois au démarrage du programme ---

// ----- Initialisation fonctionnalités utilisées -----

Serial.begin(115200); // Initialise connexion Série

//---- initialise la connexion Ethernet avec l'adresse MAC du module Ethernet, l'adresse IP Locale
//---- +/- l'adresse IP du serveurDNS , l'adresse IP de la passerelle internet et le masque du réseau local

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - utilise plus de mémoire Flash (env + 6Ko)
Ethernet.begin(mac, ipLocal); // forme conseillée pour fixer IP fixe locale
//Ethernet.begin(mac, ipLocal, serverDNS, passerelle, masque); // forme complète

delay(1000); // donne le temps à la carte Ethernet de s'initialiser

Serial.print(F("Shield Ethernet OK : L'adresse IP du shield Ethernet est : " ));

Serial.println(Ethernet.localIP());

//---- initialise le serveur ----
serveurHTTP.begin();
Serial.println(F("Serveur Ethernet OK : Ecoute sur port 80 (http)"));

} // fin de la fonction setup()
```

Fonction **loop()** (1) : Réception des caractères en provenance du client distant

Déclaration d'un objet client

- On commence par créer un objet **EthernetClient** qui sera local à la boucle **loop()** : ce client existera seulement si une connexion entrante existe, ce qui est testé à l'aide de la fonction **.available()** de l'objet **EthernetServer** précédemment configuré.

Réception des caractères

- Ensuite, si le client existe, après avoir affiché quelques messages,...
- on teste si le client est connecté : ceci est testé à l'aide de la fonction **.connected()** de l'objet **EthernetClient**.
- Puis, à l'aide d'une boucle **while()** et de la fonction **.available()** de l'objet **EthernetClient**, qui bouclera tant qu'un caractère sera présent : on affiche le caractère reçu et on l'ajoute à une chaîne de réception
- Une condition permet d'éviter la surcharge en réception au delà de 100 caractères.

```
void loop(){ // debut de la fonction loop()

// crée un objet client basé sur le client connecté au serveur
EthernetClient client = serveurHTTP.available();

if (client) { // si l'objet client n'est pas vide
// le test est VRAI si le client existe

// message d'accueil dans le Terminal Série
Serial.println (F("-----"));
Serial.println (F("Client present !"));
Serial.println (F("Voici la requete du client:"));

////////// Réception de la chaine de la requete //////////

//-- initialisation des variables utilisées pour l'échange serveur/client
chaineRecue=""; // vide le String de reception
comptChar=0; // compteur de caractères en réception à 0

if (client.connected()) { // si le client est connecté

////////// Réception de la chaine par le réseau //////////
while (client.available()) { // tant que des octets sont disponibles en lecture
// le test est vrai si il y a au moins 1 octet disponible

char c = client.read(); // l'octet suivant reçu du client est mis dans la variable c
comptChar=comptChar+1; // incrémente le compteur de caractère reçus

Serial.print(c); // affiche le caractère reçu dans le Terminal Série

//--- on ne mémorise que les n premiers caractères de la requete reçue
//--- afin de ne pas surcharger la RAM et car cela suffit pour l'analyse de la requete
if (comptChar<=100) chaineRecue=chaineRecue+c; // ajoute le caractère reçu au String pour les N premiers caractères
//else break; // une fois le nombre de caractères dépassés sort du while

} // --- fin while client.available = fin "tant que octet en lecture"

Serial.println (F("Reception requete terminee"));
```

Fonction **loop()** (2) : Affichage et analyse de la chaîne reçue

- Ensuite, tout simplement, on affiche la chaîne reçue
- Puis on teste si la chaîne commence bien l'entête GET attendue dans le cas de la réception d'une requête HTTP valide :

```
//////////////////// Affichage de la requete reçue //////////////////////
Serial.println(F("----- Affichage de la requete recue -----")); // affiche le String de la requete
Serial.println (F("Chaîne prise en compte pour analyse : "));
Serial.println(chaineRecue); // affiche le String de la requete pris en compte pour analyse

//////////////////// Analyse de la requete reçue //////////////////////
Serial.println(F("----- Analyse de la requete recue -----")); // analyse le String de la requete

//----- analyse si la chaîne reçue est une requete GET -----
if (chaineRecue.startsWith("GET")) { // si la chaîne recue commence par GET

    Serial.println (F("Requete HTTP valide !"));

//----- +/- extraction et analyse de la sous-chaîne utile -----
```

Fonction **loop()** (3) : Envoi de la réponse Http

- on envoie ensuite une réponse HTTP valide, affichée également en copie dans le terminal série :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
////////// Réponse HTTP suivie de la Page HTML de réponse //////////  
  
//-- envoi de la réponse HTTP ---  
client.println(F("HTTP/1.1 200 OK")); // entete de la réponse : protocole HTTP 1.1 et exécution requete réussie  
client.println(F("Content-Type: text/html")); // précise le type de contenu de la réponse qui suit  
client.println(F("Connection: close")); // précise que la connexion se ferme après la réponse  
client.println(); // ligne blanche  
  
//--- envoi en copie de la réponse http sur le port série  
Serial.println(F("La reponse HTTP suivante est envoyee au client distant :"));  
Serial.println(F("HTTP/1.1 200 OK"));  
Serial.println(F("Content-Type: text/html"));  
Serial.println(F("Connection: close"));  
Serial.println();
```

Remarque technique :

Noter l'utilisation abondante de la forme `println(F(« chaine »))` qui a pour effet de stocker les chaînes de caractères directement dans la mémoire programme Flash au lieu de les placer dans la RAM dont la taille est limitée : **cette façon de faire est INDISPENSABLE dès que l'on utilise de nombreuses chaînes de caractères** dans un code sous peine de bloquer l'exécution par saturation de la Ram de l'Arduino.

Je rappelle ici que l'Arduino dispose de 3 mémoires : la Ram (2Ko), la mémoire programme Flash (30Ko) et l'Eeprom de petite taille.

Fonction **loop()** (4) : Envoi du début et du head de la page HTML de réponse, **incluant le code javascript**

- Par un jeu de **println()**, on envoie la page HTML avec :
 - les balises **<html>** et **</html>** de début et fin de page
 - les balises **<head>** et **</head>** d'entête
 - **<body>** et **</body>** du corps de la page
- **A ce niveau, nous insérons la balise script et nous insérons à l'aide println successifs le code javascript vu précédemment .** Remarquer l'utilisation de la fonction **window.onload()** : il sera ainsi lancé au chargement de la page et ce qui permet de le placer dans le head bien que le script puisse potentiellement utiliser des éléments placés dans le body.

```
//--- la réponse HTML à afficher dans le navigateur
//----- début de la page HTML -----
client.println(F("<!DOCTYPE html>"));
client.println(F("<html>"));

//----- head = entete de la page HTML -----
client.println(F("<head>"));

client.println(F("<meta charset=\"utf-8\" />")); // fixe encodage caractères - utiliser idem dans navigateur
client.println(F("<title>Titre</title>")); // titre de la page HTML

//===== bloc de code javascript =====
client.println(F("<!-- Début du code Javascript -->"));
client.println(F("<script language=\"javascript\" type=\"text/javascript\">"));
client.println(F("<!--      \""));

client.println(F("window.onload = function () { // au chargement de la page"));

client.println(F("var canvas = document.getElementById(\"nomCanvas\"); // declare objet canvas a partir id = nom "));
client.println(F("canvas.width = 300; // largeur canvas"));
client.println(F("canvas.height = 300; // hauteur canvas"));

client.println(F("if (canvas.getContext){ // la fonction getContext() renvoie True si canvas accessible"));

    client.println(F("var ctx = canvas.getContext(\"2d\"); // objet context permettant acces aux fonctions de dessin"));
    client.println(F("ctx.fillStyle = \"rgb(200,200,200)\"; // couleur de remplissage rgb 0-255"));
    client.println(F("ctx.fillRect (0, 0, canvas.width, canvas.height); // rectangle "));
    client.println(F("ctx.fillStyle = \"rgb(0,255,0)\"; // couleur remplissage"));
    client.println(F("ctx.fillRect (50, 50, 200, 200); // rectangle"));

client.println(F("} // fin si canvas existe"));

client.println(F("else {"));
client.println(F("// code si canvas non disponible "));
client.println(F("} // fin else"));

client.println(F("} // fin onload"));

client.println(F("//-->"));
client.println(F("</script>"));
client.println(F("<!-- Fin du code Javascript --> "));
//===== fin du bloc de code javascript =====

client.println(F("</head>"));
//----- fin head = fin entete de la page HTML -----
```

Fonction **loop()** (5) : envoi du body et de la fin de la page HTML de réponse

- De la même façon, par un jeu de `println()`, on envoie la suite du code HTML au besoin.
- **Ici, point essentiel, on insère la balise canvas dans le body de la page HTML.**
- Suivent les balises de clôture de la page web

```
//----- body = corps de la page HTML -----
client.println("<body>");

// affiche chaines caractères simples
client.println("<CENTER>"); //pour centrer la suite de la page
client.println("<canvas id=\"nomCanvas\" width=\"300\" height=\"300\"></canvas>");
client.println("<br/>");
client.println("<br/>");
client.println("Serveur Arduino : Test de page HTML incluant un canvas");
client.println("<br/>");

client.println("</body>");
//----- fin body = fin corps de la page -----

client.println("</html>");
//----- fin de la page HTML -----

} // fin if GET
```


Fonction **loop()** (6) : Fermeture de la connexion avec le client

- si la requête reçue n'est pas une « GET », un message indique que la requête n'est pas valide.
- Puis la connexion avec le client est clotûrée.

```
    else { // si la chaine recue ne commence pas par GET
      Serial.println (F("Requete HTTP non valide !"));
    } // fin else

    //----- fermeture de la connexion -----

    // fermeture de la connexion avec le client après envoi réponse
    delay(1); // laisse le temps au client de recevoir la réponse
    client.stop();
    Serial.println(F("----- Fermeture de la connexion avec le client -----")); // affiche le String de la requete
    Serial.println (F(""));

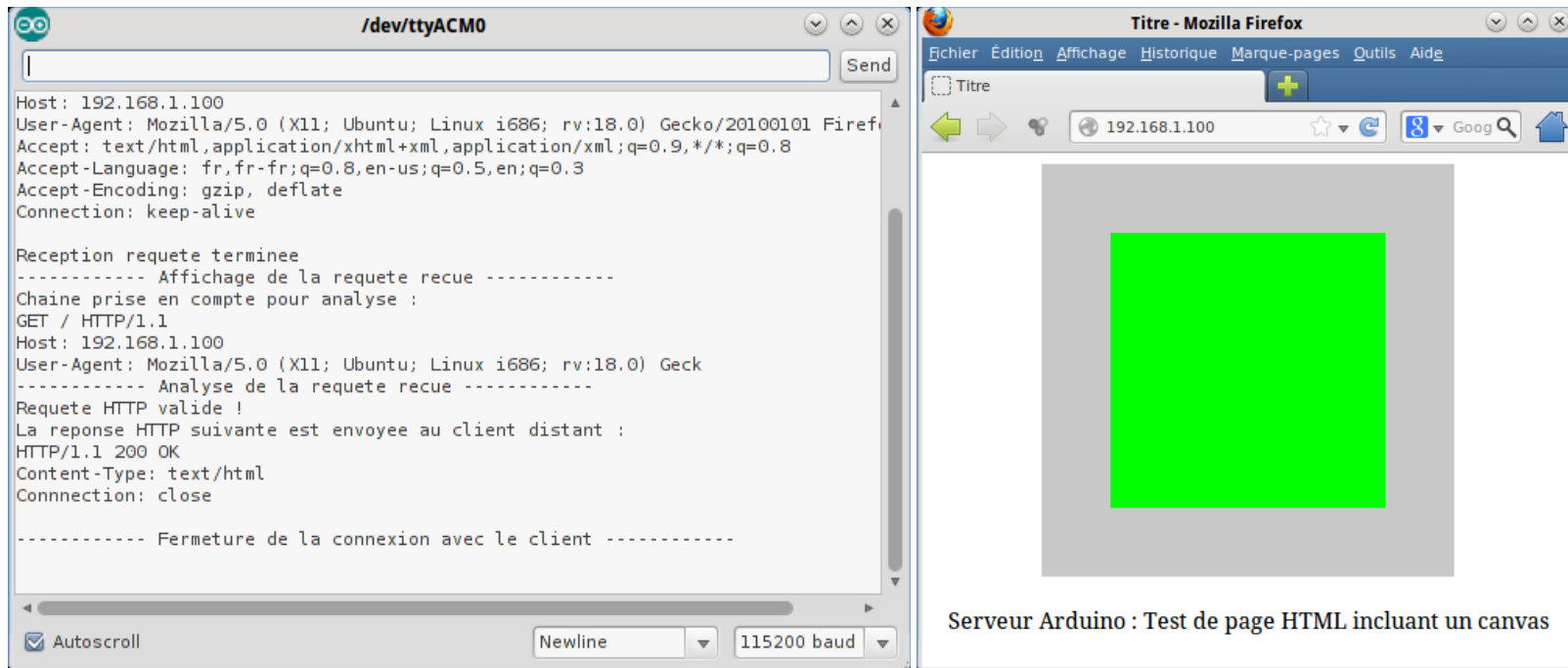
    } // --- fin if client connected

  } //---- fin if client ----

} // fin de la fonction loop()
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80)
- A présent, **ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse** (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Dans le navigateur, on doit ici voir notre canva s'afficher avec un carré vert :



Le canvas s'affiche dans le navigateur ? Bravo ! Vous avez écrit votre premier code serveur Arduino utilisant un canvas et un code Javascript !
Les bases pour la mise en place d'interfaces graphiques côté navigateur client sont ainsi posées !!

Synthèse

Dans ce programme, du côté Arduino : la génération de la page HTML et l'envoi du code Javascript à exécuter par le navigateur client.

Du côté navigateur client : affichage de la page HTML et exécution du code Javascript inséré dans la page HTML

Au final, il suffit de se connecter à l'aide d'un navigateur à l'adresse IP du shield Ethernet sans aucune configuration particulière du côté du client pour que celui-ci exécute le code javascript envoyé par le serveur Arduino : simple et efficace !

25. Les éléments du langage Arduino étudiés dans cet atelier

Les fonctions de la librairie Ethernet

Chaque classe dispose de plusieurs fonctions associées :

Classe *Ethernet* (configuration matérielle du shield Ethernet)

- | begin() | localIP() | maintain()

Classe *EthernetServer* (serveur TCP)

- | begin() | available() | write() | print() | println()

Classe *EthernetClient* (client TCP)

- | connected() | connect() | write() | print() | println() | available() | read() | flush() | stop()

La documentation complète du langage Arduino en français est disponible ici :
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi

Pratique :
Les codes de cet atelier sont disponibles ici :
http://www.mon-club-elec.fr/mes_downloads/tutos_arduino/12b3.atelier_arduino_ethernet_tcp_javascript_canva.tar.gz

26. *A présent, vous devriez être capable :*

- d'écrire un programme javascript inséré dans une page HTML
- d'écrire un programme javascript utilisant un canvas
- d'écrire un programme Arduino de serveur de page HTML + Javascript

Table des matières

Créer un serveur HTML+Javascript avec Arduino et afficher un canvas (zone de dessin) dans le navigateur client.

Intro |
Matériel nécessaire pour les ateliers Arduino |
Matériel spécifique nécessaire pour cet atelier |
Matériel spécifique nécessaire pour cet atelier (suite) |
Rappel : structure d'un réseau local et matériel nécessaire |
Un peu de vocabulaire pour avoir les idées claires |
La structure du réseau que nous allons réaliser |
Les éléments du réseau local que nous allons utiliser |
Le shield Ethernet : description et principe d'utilisation |
Monter le réseau utilisant le shield Ethernet Arduino |
Monter le réseau utilisant le shield Ethernet Arduino sur un réseau avec « box » existant |
Rappel : Structure type d'une page HTML simple et écrire une première page HTML |
Notion de script « côté serveur » et de page web « dynamique » |
Notion de script « côté client » et présentation du langage Javascript |
Syntaxe de base du langage Javascript |
Ecrire un premier script Javascript intégré dans une page HTML |
Principe de test et d'exécution d'une page HTML incluant un script javascript |
Serveur Arduino Tcp : un premier code utilisant un code javascript |
Pour info : le DOM, l'accès aux éléments d'une page HTML et les fonctions de l'objet window |
HTML : Présentation de l'objet canvas |
Javascript : Les fonctions essentielles de l'objet canvas |
Exemple de page HTML+ Javascript : dessiner dans un canvas |
Serveur Arduino : Envoyer une page HTML + Javascript et afficher un canvas dans le navigateur client |
Les éléments du langage Arduino étudiés dans cet atelier |
A présent, vous devriez être capable : |

Bravo !
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :
http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS