Résumé Première NSI

7. Les réseaux

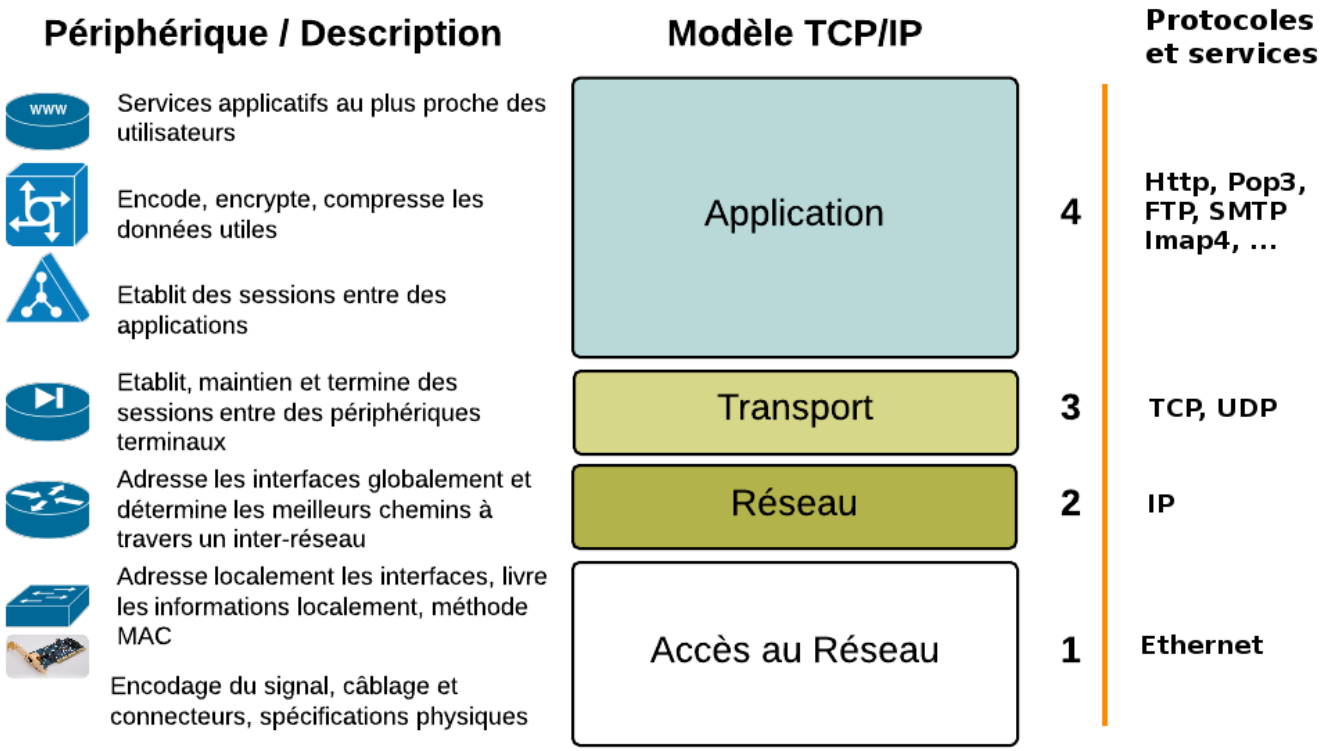
Communication

ARPANET : Le 29 octobre **1969**, le réseau Arpanet effectuait sa première communication. Les trois dernières lettres de ce premier message, le simple mot "login", mettent plusieurs heures avant de parvenir à destination mais qu'importe, le *Advanced Research Projects Agency Network*, ou ARPAnet, a rempli sa première mission : des paquets de données ont transité avec succès entre l'université de Californie à Los Angeles (UCLA) et l'Institut de recherche de Stanford au sud de San Francisco. L'ancêtre d'internet est né ! Et il s'agit d'une collaboration entre militaires et universitaires américains.

Vinton Cerf pendant son cursus universitaire, commence à travailler sur le projet Arpanet, le premier réseau de transmission de données par paquets, avant d'être nommé professeur à Stanford, où il imagine, avec **Bob Kahn**, un moyen de relier différents réseaux baptisé « internetwork », qui est formalisé **en 1974** dans l'article *A Protocol for Packet Network Intercommunication*. C'est l'acte de **naissance du protocole TCP/IP**.

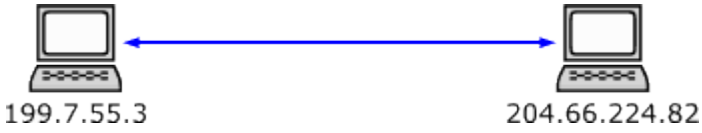
Eléments de base

Le modèle TCP/IP est en quatre couches :



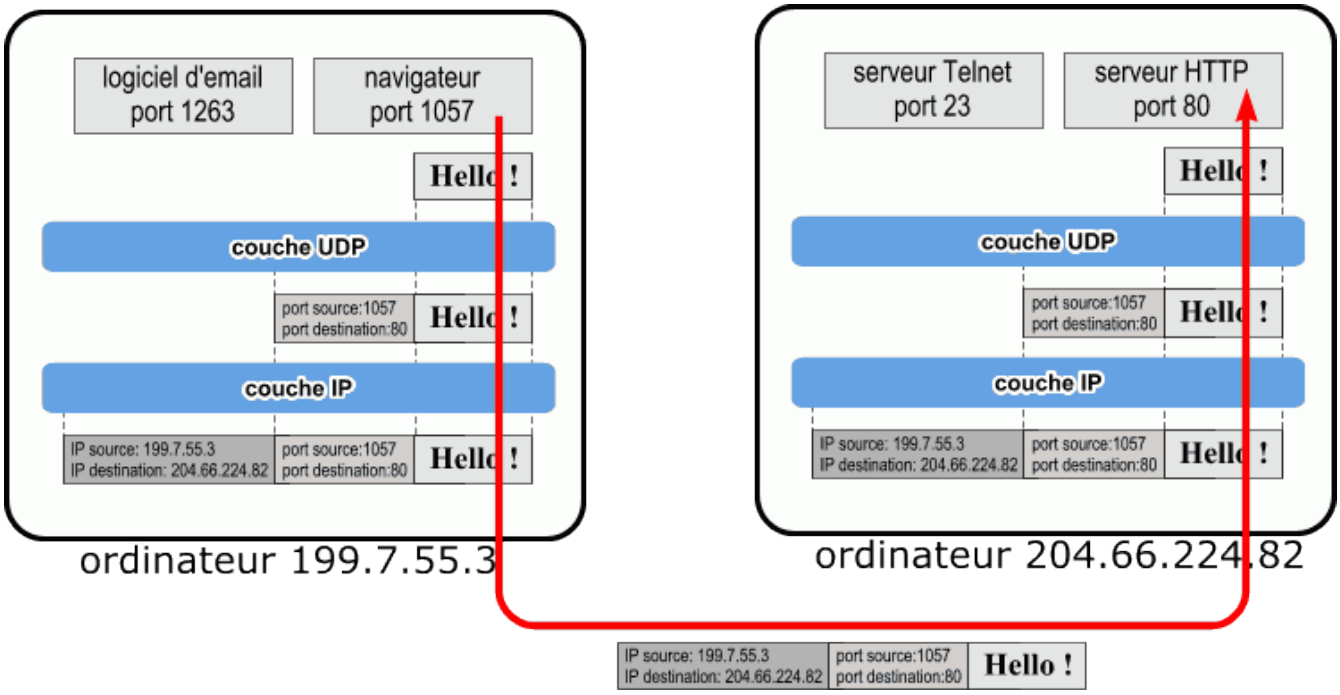
Rôle des différents protocoles

UDP/IP est un protocole qui permet d'utiliser des numéros de **ports** en plus des **adresses IP** (On l'appelle UDP/IP car UDP fonctionne au dessus d'IP). IP s'occupe des adresses IP et UDP s'occupe des ports. Avec le protocole **IP** on peut envoyer des données d'un ordinateur A à un ordinateur B.



Avec **UDP/IP**, on peut être plus précis : on envoie des données d'une **application x** sur l'**ordinateur A** vers une **application y** sur l'**ordinateur B**.

Par exemple, votre navigateur peut envoyer une requête à un serveur HTTP (un serveur Web):



Chaque couche (UDP et IP) va ajouter ses informations.
Les informations de **IP** vont permettre d'acheminer le paquet à destination du bon **ordinateur**. Une fois arrivé à l'ordinateur en question, la couche **UDP** va délivrer le paquet au bon **logiciel** (dans l'exemple ici : au serveur HTTP).

- Les deux logiciels se contentent d'émettre et de recevoir des données ("**Hello !**"). Les couches UDP et IP en dessous s'occupent de tout.

Ce couple (199.7.55.3:1057, 204.66.224.82:80) est appelé un **socket**. Un socket identifie de façon unique une communication entre deux logiciels.

TCP/IP

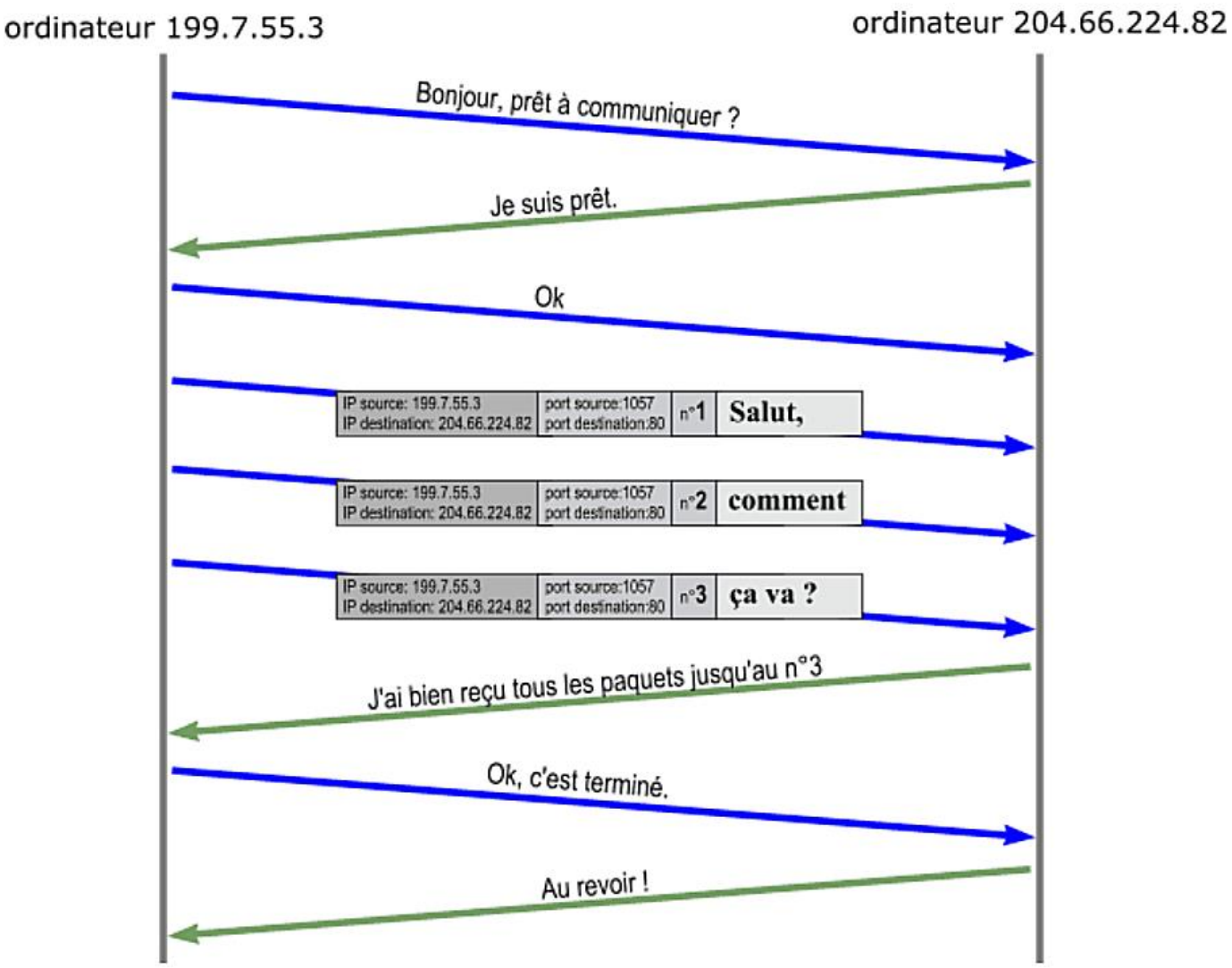
Bon... on peut maintenant faire communiquer 2 logiciels situés sur des ordinateurs différents. Mais il y a encore de petits problèmes :

- Quand vous envoyez un paquet IP sur Internet, il passe par des dizaines d'ordinateurs. Et il arrive que des paquets IP se perdent ou arrivent en double exemplaire.
  Ça peut être gênant : imaginez un ordre de débit sur votre compte bancaire arrivant deux fois ou un ordre de crédit perdu !
- Même si le paquet arrive à destination, rien ne vous permet de savoir si le paquet est bien arrivé (aucun accusé de réception).
- La taille des paquets IP est limitée (environ 1500 octets).
  Comment faire pour envoyer la photo JPEG qui fait 62000 octets ?
- C'est pour cela qu'a été conçu TCP.

TCP est capable :

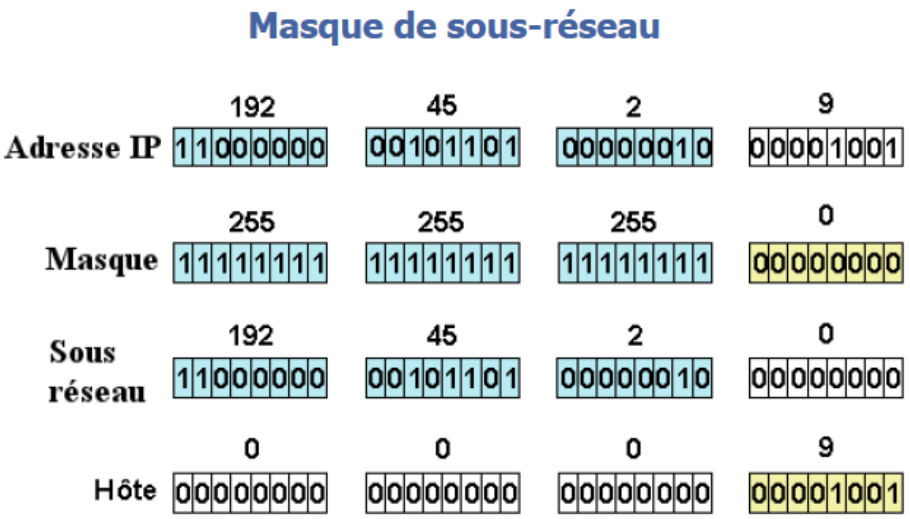
- de faire tout ce que UDP sait faire (il gère les numéros de ports).
- de vérifier que le destinataire est prêt à recevoir les données.
- de découper les gros paquets de données en paquets plus petits pour que IP les accepte.
- de numéroter les paquets, et à la réception de vérifier qu'ils sont tous bien arrivés, de redemander les paquets manquants et de les réassembler avant de les donner aux logiciels. Des accusés de réception sont envoyés pour prévenir l'expéditeur que les données sont bien arrivées.

Par exemple, pour envoyer le message "Salut, comment ça va ?", voilà ce que fait TCP (Chaque flèche représente un paquet IP):



- A l'arrivée, sur l'ordinateur 204.66.224.82, la couche TCP reconstitue le message "Salut, comment ça va ?" à partir des 3 paquets IP reçus et le donne au logiciel qui écoute le port 80.

Adresse IP :
Dans IPv4 l'adresse est constituée de 4 octets. La première partie sert comme identifiant du réseau local.
La deuxième partie sert comme identifiant de la machine.



Le masque de sous réseau permet de séparer dans une adresse IP la partie réseau de la partie hôte (machine)
Remarques :

- Si le masque de sous réseau est 255.255.255.0, alors tous les terminaux appartenant au même sous réseau auront les 3 premiers octets de leur adresse IP identiques. Dans le cas ci-dessus (en exemple) elles commenceront toutes par 192.45.2
- A la suite de l'adresse IP on ajoute /24 (3 octets soit 24 bits déterminent la partie réseau de l'adresse)
- 192.45.2.9/24 et 192.45.2.11/24 appartiennent au même sous réseau, en revanche 192.45.6.24/24 n'appartient pas au sous réseau

Les jours du protocole IP dans sa forme actuelle (IPv4) sont comptés. Le réseau Internet était utilisé largement par les universités, les industries de pointe, et le gouvernement dès le milieu des années 1990, mais Internet intéresse de plus en plus les entreprises et les sociétés commerciales - il sera utilisé par un grand nombre d'individus et de systèmes exprimant des besoins différents. Le protocole **IPv6** (appelé également **IPng** pour *IP new generation*) offre plus de flexibilité et d'efficacité, résout toute une variété de problèmes et ne devrait jamais être en rupture d'adresses.

La notation IPv6

Une nouvelle notation a été définie pour décrire les adresses IPv6 de **16 octets**. Elle comprend 8 groupes de 4 chiffres hexadécimaux séparés avec le symbole deux-points. Par exemple :

8000:0000:0000:0000:0123:4567:89AB:CDEF

- Puisque plusieurs adresses ont de nombreux zéros dans leur libellé, 3 optimisations ont été définies. Tout d'abord, les premiers zéros d'un groupe peuvent être omis, comme par exemple 0123 qui peut s'écrire 123. Ensuite, un ou plusieurs groupes de 4 zéros consécutifs peuvent être remplacés par un double deux-points. C'est ainsi que l'adresse ci-dessus devient :

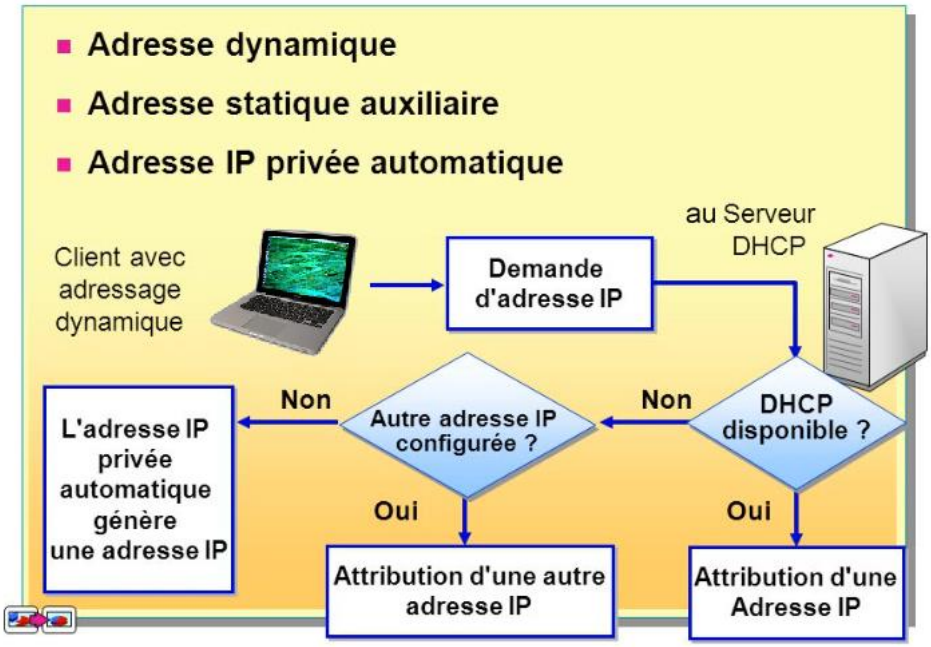
8000:::123:4567:89AB:CDEF

- Enfin, les adresses IPv4 peuvent être écrites en utilisant la représentation de l'adresse en notation décimale pointée précédée d'un double deux-points, comme par exemple :

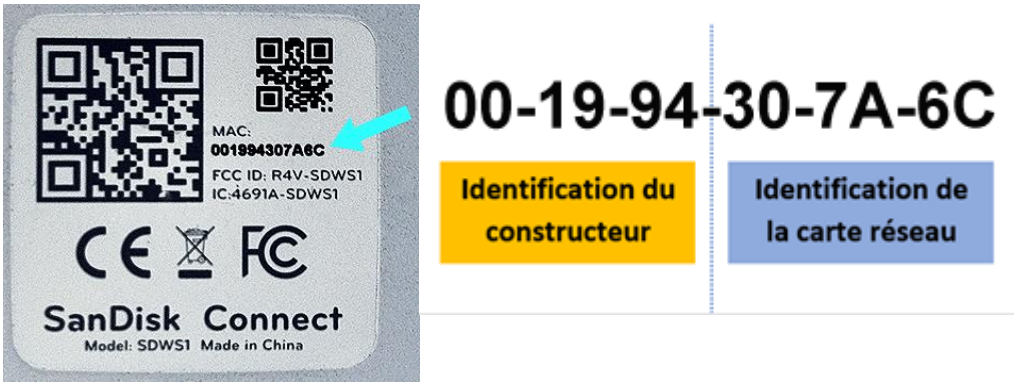
::192.31.254.46

Architecture d'un réseau

- Sur un réseau local, une machine peut avoir une **adresse IP statique** (permanente) ou bien être configurée pour demander au serveur DHCP (Dynamic Host Configuration Protocol) une adresse IP qui change chaque fois qu'elle se connecte à ce réseau. Dans ce cas on dit que c'est une **adresse IP dynamique**.



- Chaque matériel connecté à un réseau a au moins une carte électronique réseau. Celle-ci est identifiée par une adresse MAC (6 octets). Les 3 premiers octets identifient le constructeur. Exemple :



Quelques commandes Linux en rapport avec le réseau :

- **ifconfig** est utilisé pour configurer les paramètres d'interface réseau.

Exemple : `$ ifconfig`
renvoie :

```
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:d0:24:55  txqueuelen 1000  (Ethernet)…/…

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)…/…

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.12  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::5308:ae80:e801:d4d6  prefixlen 64  scopeid 0x20<link>
        inet6 2a01:cb05:8315:cb00:db98:d1f6:d6a0:e98  prefixlen 64  scopeid …/…
```

- **ping** permet de tester la connectivité de la machine au réseau.

`ping <adresseIP_de_la_machine_distante>` ou `ping <adresse_url>` (on arrête avec Ctrl + c).

Exemple : `$ ping education.gouv.fr`

renvoie :

```
PING education.gouv.fr (185.75.143.24) 56(84) bytes of data.
64 bytes from MEN-WEBEDU-PROXY01.dedie.ate.info (185.75.143.24): icmp_seq=1 ttl=56 time=17.6 ms
64 bytes from MEN-WEBEDU-PROXY01.dedie.ate.info (185.75.143.24): icmp_seq=2 ttl=56 time=17.7 ms
64 bytes from MEN-WEBEDU-PROXY01.dedie.ate.info (185.75.143.24): icmp_seq=3 ttl=56 time=17.5 ms
64 bytes from MEN-WEBEDU-PROXY01.dedie.ate.info (185.75.143.24): icmp_seq=4 ttl=56 time=17.8 ms
64 bytes from MEN-WEBEDU-PROXY01.dedie.ate.info (185.75.143.24): icmp_seq=5 ttl=56 time=17.8 ms
--- education.gouv.fr ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 17.520/17.712/17.843/0.151 ms
```

Remarque : le nom identifiant la machine hôte `MEN-WEBEDU-PROXY01.dedie.ate.info` est retourné.

- **traceroute** affiche le chemin emprunté par les paquets pour l'hôte réseau.

`traceroute <adresseIP_de_la_machine_distante>` ou `traceroute <adresse_url>`

Exemple : `$ traceroute education.gouv.fr`

renvoie la route qu'un paquet prend pour atteindre la machine hôte distante.

- La première colonne donne le numéro du saut à travers un routeur sur internet.
- La deuxième colonne donne l'adresse IP de ce routeur et après on voit trois durées en millisecondes . traceroute envoie 3 paquets à ce routeur et écrit chaque durée mise par ces paquets pour l'atteindre.

```
traceroute to education.gouv.fr (185.75.143.24), 30 hops max, 60 byte packets
 1  livebox.home (192.168.1.1)  4.059 ms  4.181 ms  4.135 ms
 2  * * *
 3  lag-116.ncnan101.Nantes.francetelecom.net (193.253.151.6)  8.404 ms  8.551 ms  8.515 ms
 4  193.252.162.242 (193.252.162.242)  14.758 ms  14.873 ms  14.836 ms
 5  ae41-0.nrlil101.Villeneuve-dascq.francetelecom.net (193.252.160.217)  18.907 ms  19.388 ms  19.657 ms
 6  lag-1.nolil601.Villeneuve-dascq.francetelecom.net (193.252.100.94)  19.632 ms  16.479 ms  16.724 ms
 7  193.252.227.62 (193.252.227.62)  16.751 ms  16.365 ms  16.208 ms
 8  core01-civ_core01-ate.as35625.net (185.39.168.25)  18.649 ms  16.416 ms  16.326 ms
```

- **getent hosts** permet d'obtenir l'adresse IP correspondant à un nom de domaine.

`getent hosts <nom de domaine>`

Exemple : `$ getent hosts education.gouv.fr`

renvoie : `185.75.143.24  education.gouv.fr`

`getent hosts` permet aussi de connaitre le nom de la machine associée à une adresse IP :

Exemple : `$ getent hosts 185.75.143.24`

Renvoie : `185.75.143.24  MEN-WEBEDU-PROXY01.dedie.ate.info`

Une instruction Python en rapport avec le réseau :

- **socket.gethostbyaddr** permet aussi d'obtenir le nom de la machine et l'adresse IP à partir d'une URL.

```
import socket
print(socket.gethostbyaddr("nom de domaine"))
```

Exemple :

```
import socket
print(socket.gethostbyaddr("education.gouv.fr"))
```

renvoie : `('MEN-WEBEDU-PROXY01.dedie.ate.info', [], ['185.75.143.24'])`

Exemple :

```
import socket
print(socket.gethostbyaddr("185.75.143.24"))
```

renvoie : `('MEN-WEBEDU-PROXY01.dedie.ate.info', [], ['185.75.143.24'])`

8. **Le web** – Remarque : la numérotation des paragraphes n'est pas logique, mais elle reprend celle du cours.

8.1 Le langage HTML

- Le langage HTML *HyperText Markup Language* est un langage de description d'une page Web. Ce n'est pas un langage de programmation.
- Un navigateur tel que Edge, Safari, Chrome, FireFox interprète les ligne de code HTML
- www ou world wide web est la partie d'Internet qui regroupe toutes les pages en langage HTML reliées entre elles par des liens hypertexte.
- Le langage HTML utilise des balises (*tags* en anglais) écrites entre les symboles < et >. La plupart des balises sont composées d'une balise ouvrante <…> et d'une balise fermante </…>.
- Quelques balises sont orphelines c'est-à-dire seules. Exemples : <meta>,
, <img> etc.

8.1.1 à 8.1.4 Création d'une page, création d'autres pages, les images, les balises HTML

- La structure minimale d'une page web est constituée d'une balise **<html>** et de deux sections :

  <head> (l'en-tête de la page) et **<body>** (le corps de la page) .

- En tout première ligne <! **DOCTYPE html**> est une déclaration qui informe le navigateur d'un visiteur que ce document est écrit en HTML. Ce n'est pas à proprement parler une balise HTML.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Mon titre</title>
    </head>

    <body>

    </body>
</html>
```

Commentaires	`<!--`
	`Ceci est un commentaire`
	`-->`

Lien hypertexte vers une boite mail	`<a href="#mailto.jdubois@avesnieres.fr">Mon mel</a>`
Lien vers une page d'un autre site	`<a href="https://avesnieres.paysdelaloire.e-lyco.fr"> Mon lycée </a>`
Placer un identifiant dans une balise	`<h1 id="haut_de_page"> Le premier paragraphe </h1>`
Lien hypertexte vers une balise qui possède un identifiant	`<a href="#haut_de_page"> Cliquer ici pour aller en haut </a>`
Insérer une image	`<img src="ma_photo.jpeg" alt="ma photo">`

Insérer une image avec une légende (*caption* en anglais)	`<figure>`
	`  <img src="lpa-logo.png" alt="Logo du lycée">`
	`  <figcaption> Lycée d'Avesnières </figcaption>`
	`</figure>`

8.1.5 Le langage CSS

Il permet de séparer les éléments de mise en forme du contenu du document HTML.

1ère méthode :
dite « inline CSS »

Insérer **un attribut** `style=` **dans une balise**

```
<h2 style="text-decoration:underline;color:blue"> Mon titre </h2>
<p style="color:green"> Mon paragraphe en vert </p>
```

Utilisation de la balise
span signifie portée.

Exemple : pour appliquer le style texte en italiques sur une portée limitée à l'intérieur d'un paragraphe :

```
<h3 style="color:red"><span style="font-style:italic">Je me</span> présente</h3>
```

donne :
Je me présente

2ème méthode :
dite « internal CSS »

Ajouter **dans la section <head>** des règles CSS sous la forme d'une **balise <style>**

```
<style type='text/css'>
…
</style>
```

```
<head>
    <meta charset="utf-8" />
    <style type="text/css">
        h2{text-decoration:underline;color:blue}
        p{color:green}
    </style>
    <title>Mon titre</title>
</head>
```

3ème méthode :
dite « external CSS »

1) Créer un fichier de style nommé style.css

2) Ajouter **dans la section <head>** de chaque page une **balise link**

```
<link rel='stylesheet' href=..>
```

1) Créer une feuille de style séparée nommée style.css ***placée dans le même dossier***

```
h2{
    text-decoration: underline;
    color: blue;
}

p{
    color: green;
}
```

2) Ajouter une balise <link> dans la section <head> des pages HTML qu'on veut relier à cette feuille de style :

```
<head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <title>Mon titre</title>
</head>
```

Utilisation de la balise

div signifie division.

Dans la page HTML :

```
<body>
    <!-- Début de l'en-tête -->
    <div id="en-tete">
        <h1> Jacques DUBOIS </h1>
    </div>

    <!-- Début du menu -->
    <div id="menu">
        <ul>
            <li>Accueil</li>
            <li>Ville</li>
            <li>Passion</li>
        </ul>
    </div>

    <!-- Début de la présentation -->
    <div id="presentation">
        <h3>Je me présente :</h3>
        <p>Je m'appelle ...</p>
    </div>

    <!!-- Début du pied de page -->
    <div id="pied">
        <img src="...">
    </div>
</body>
```

Dans le fichier style.css placé dans le même dossier

```
/* Mon css */
body {  margin : 10px;
    background-color : #F8F8FF;
    font-family : Arial;
}

h3 {    font-size : 30px;
    text-align : left;
}

p {     text-align : justify;
    color : #003366;
}

#en-tete { background-color : #99CCCC;
    padding : 10px 10px 0px 10px;
}

#menu { float : left;
    background-color : #FFFFFF;
}

#presentation {
    margin-left : 270px;
    padding : 10px 10px 50px 0;
    color : #003366;
}

#pied { background-color:#99CCCC;
    text-align:right;
}
```

Ajout d'un attribut class dans une balise

Pour définir un style pour un type spécial d'éléments, on ajoute un attribut class à cet élément.

Exemple :

```
<img src="mon_village.jpg"  class="imageflottante">
```

Définition de cette classe dans style.css

```
.imageflottante{
    float:left;
}
```

8.2.3 Le langage de programmation JavaScript

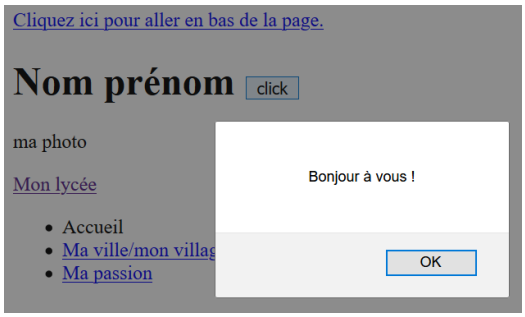
JavaScript est langage de programmation qui permet de faire agir l'utilisateur sur une page HTML.

Dans la section <head>, l'élément `<script>` contient *soit* des fonctions de script, *soit* il pointe vers un fichier extérieur précisé avec l'attribut `src`. **Dans la section <body>** un événement sur un objet exécute la fonction.

1. Exemples où **<script> contient les instructions** JavaScript

8.1.6 Exemples avec les trois sortes de boites de dialogue JavaScript `alert()`, `confirm()` et `prompt()`.

1ère sorte : boite de type « alert »
Elle *affiche* un message et possède un bouton « OK »

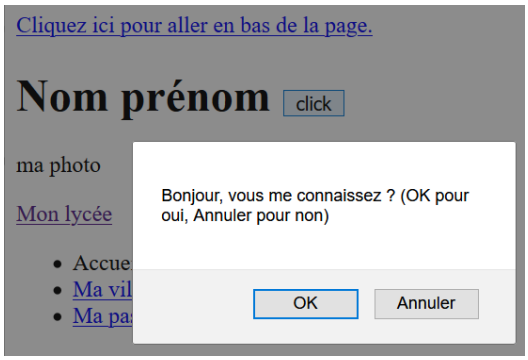


C'est l'instruction JavaScript `alert('texte affiché')` qui permet d'afficher la boite.
Puisque dans cet exemple on a placé l'instruction `alert()`dans la fonction bonjour() il sera nécessaire d'exécuter cette fonction pour faire apparaitre sur la boite d'alerte.
Dans l'exemple, elle est exécutée lorsque survient l'évènement clic de souris sur l'élément « bouton » inséré à l'aide d'une balise <button> </button>.

```
    <script type="text/javascript">
        function bonjour(){
            alert('Bonjour à vous !');
            }
    </script>
</head>

<body>
        <!-- Début de l'en-tête -->
    <div id="en-tete">
        <a href="#bas_de_page"> Cliquez ici pour a
ller en bas de la page. </a>
        <h1 id="haut_de_page"> Nom prénom <button
onclick='bonjour()'>click</button></h1>
```

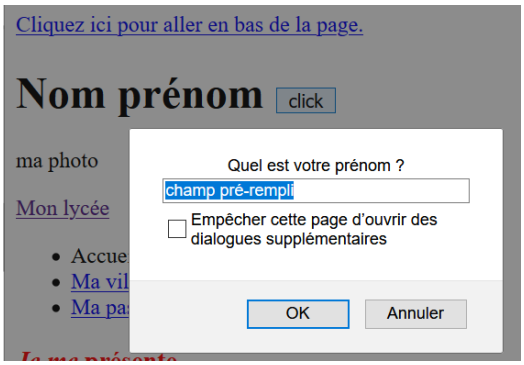
2ème sorte : boite de type « confirm »
Elle *demande une confirmation*.



C'est l'instruction JavaScript `confirm('texte affiché')` qui permet d'afficher cette boite.
La boîte de dialogue confirm() affiche un message et deux boutons : « OK » et « Annuler ». Cette fonction retourne une valeur booléenne qui vaut « true » si c'est le bouton OK qui est cliqué et retourne « false » si c'est le bouton « Annuler » qui est cliqué.

```
<script type="text/javascript">
        function confirmation()  {
            if (confirm('Bonjour, vous me connaiss
ez ? (OK pour oui, Annuler pour non)')){
            /* Si l'utilisateur clique 'OK' */
            return true;
            }else{
            /* Si l'utilisateur clique 'Annule' */
            return false;
            }
        }
```

3ème sorte : boite de type « prompt »
Elle *demande une entrée au clavier*.



C'est l'instruction JavaScript `prompt('texte affiché')` qui permet d'afficher cette boite.

La boîte de dialogue prompt() permet d'afficher un message, un champ à remplir (de type texte) et deux boutons « Annuler » et « OK ».
Cette fonction retourne la valeur qui a été entrée dans le champ par l'utilisateur après un clic de souris sur « OK » ou 'null' si l'utilisateur a cliqué sur « Annuler ».

```
<script type="text/javascript">
    function confirmation()  {
        if (confirm('Bonjour, vous me connaissez ? (OK pour
 oui, Annuler pour non)')) {
           /* Si l'utilisateur clique sur 'OK' */
           return true;
           }
        else {
           /* Si l'utilisateur clique sur 'Annuler' */
           return false;
           }
    }

    function bonjour()  {
        if (confirmation())  {
        /* Si la fonction confirmation a renvoyé true */
        noma = prompt("Quel est votre prénom ?", "champ pré
-rempli");
        alert("Bonjour " + noma + " !");
        }
        else {
        /* Si la fonction confirmation a renvoyé false */
        alert("Bonjour inconnu (e) !");
        }
    }
 </script>
</head>

<body>
        <!-- Début de l'en-tête -->
    <div id="en-tete">
        <a href="#bas_de_page"> Cliquez ici pour aller en ba
s de la page. </a>
        <h1 id="haut_de_page"> Nom prénom <button onclick='b
onjour()'>click</button></h1>
```

Comment une page web est envoyée au navigateur d'un poste client par un serveur ?

1) Je clique sur un lien.
2) L'adresse URL se met dans la barre d'adresse.
3) Le protocole DNS permet d'obtenir une adresse IP à partir de l'adresse URL.
4) Le navigateur envoie sa requête au serveur.
5) Il s'agit d'une requête http envoyée au serveur.
6) Le serveur envoie le code contenu dans le fichier HTML.
7) Les paquets envoyés sont générés par le protocole TCP.
8) Les paquets circulent à l'aide du protocole IP et du protocole Ethernet.
9) Ce protocole identifie la machine émettrice et la machine réceptrice (grâce aux adresses MAC).
10) A l'arrivée les paquets sont assemblés.
11) La page est reconstituée et interprétée par le navigateur.

2. Exemples où **<script> pointe vers un fichier externe** script.js ***placé dans le même dossier***.

Fichier HTML :

```
<head>
    <title> Nom prénom </title>
    <meta charset="utf-8" />
    <link href="style.css" rel="stylesheet" type="text/css">
    <script langage="JavaScript" src="script.js"> </script>
</head>
<body onLoad="rebours(5)">
```

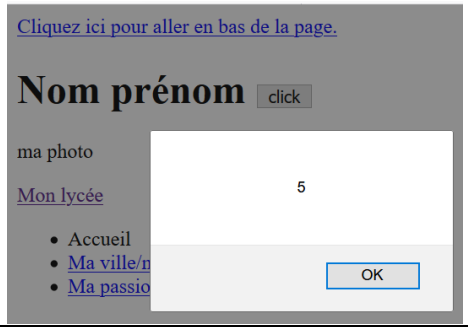
L'évènement onLoad survient lorsqu'une page web se charge sur le navigateur.

Résultat :

A l'ouverture de la page HTML apparait cette boite alert :

Ficher externe **script.js**

```
function rebours(x){
    while (x>=0){
        alert(x);
        x--
    }
}
```



Fichier HTML :

```
<head>
    <title> Nom prénom </title>
    <meta charset="utf-8" />
    <link href="style.css" rel="stylesheet" type="text/css">
    <script langage="JavaScript" src="script.js"> </script>
</head>

<body onLoad="rebours(5)">

    <!-- Début de l'en-tête -->
<div id="en-tete">
<a href="#bas_de_page"> Cliquez ici pour aller en bas de la page. </a>
<h1 onClick=message('mon\x20nom')> Henry Laval </h1>
<h2 onMouseover=message('mon\x20e-mail')> h-laval@monmel.fr </h2>
```

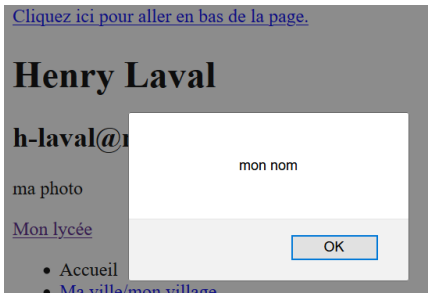
Remarque \x20 est le code ASCII hexadécimal du caractère espace.

Résultat :
En cliquant sur « Henry Laval » apparait la boite alert :

Ficher externe **script.js**

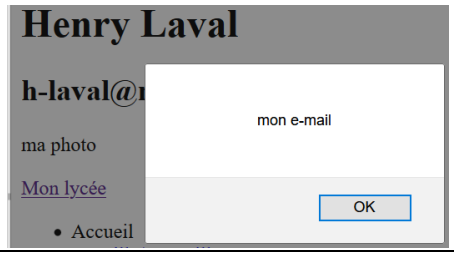
```
function rebours(x){
    while (x>=0){
        alert(x);
        x--
    }
}

function message(cause){
    alert(cause);
}
```




```
<h2 onMouseover=message('mon\x20e-mail')> h-laval@monmel.fr </h2>
```

Résultat : En passant sur la souris sur h-laval@monmel.fr apparait la boite alert :

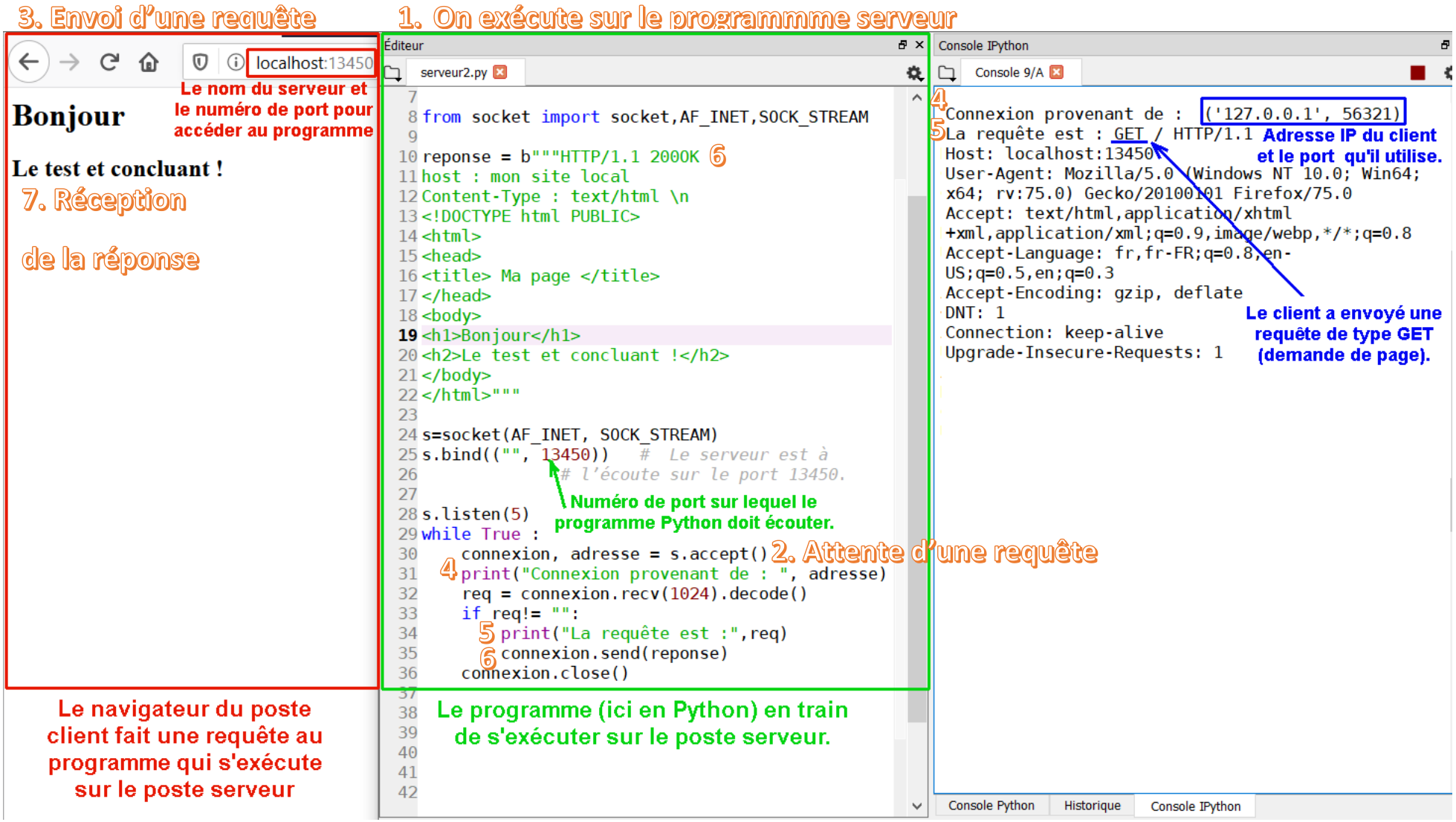


ou pour mieux délimiter la portée de la zone sensible au passage de la souris dessus, on code avec une balise :

```
<h2> <span onMouseover=message('mon\x20e-mail')> h-laval@monmel.fr </span> </h2>
```

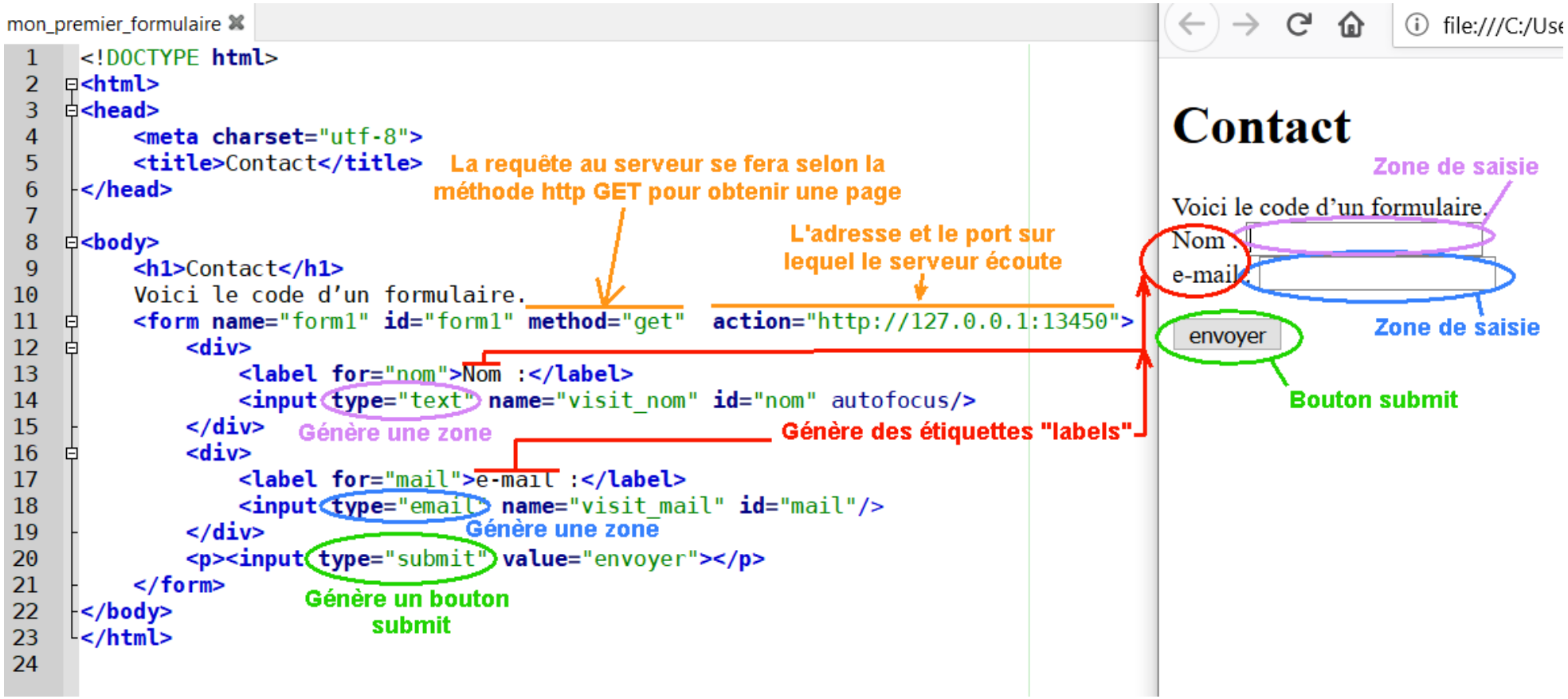
8.3 Le principe du client et du serveur – Un exemple où le logiciel client est un navigateur et le logiciel serveur est un programme Python en boucle infinie.

- Une fois qu'on a lancé le programme serveur, celui-ci écoute en permanence le port logiciel 13450 (dans cet exemple).
- Dès réception d'une requête http, l'instruction `s.accept()` est exécutée. Au final, le programme serveur renvoie la réponse HTML préparée.



8.4.1 Formulaire avec méthode GET dans une page HTML sans JavaScript

- **La méthode GET est utilisée dans le formulaire donc les données apparaitront dans l'adresse formée, après un ?**
- **Le langage de programmation JavaScript n'est pas utilisé donc il n'y aura pas de fonction telles que la vérification du contenu du formulaire avant soumission.**



1) Ce que reçoit le programme serveur **Avant que l'utilisateur appuie** sur le bouton « envoyer » dans le navigateur :

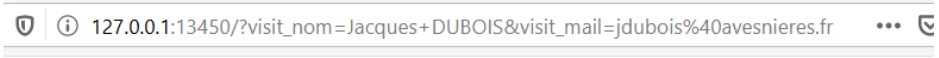

```
"""
Programme en Python sur le serveur

"""
from socket import socket, AF_INET, SOCK_STREAM
s=socket(AF_INET, SOCK_STREAM)
s.bind(("", 13450))  #  Le serveur est à l'écout
s.listen(5)
test = True
while test :
    connexion, adresse = s.accept()
    print("connexion de : ", adresse)
    req = connexion.recv(1024).decode()
    if req!= "":
        print("La requête reçue : ",req)
        connexion.send(b"Donnees bien recues !")
        if req == "Fin" :
            test = False
connexion.close()
s.close()
```

```
In [3]: runfile('C:/Users/Utilisateur/serveur1.py',
wdir='C:/Users/Utilisateur')
```

2) **Juste après avoir appuyé** sur le bouton « envoyer », la requête est envoyée au serveur avec la méthode GET. Les données *apparaissent en clair* dans la barre d'adresse du navigateur après l'URL, séparée par un point d'interrogation.

L'adresse qui se forme dans la barre d'adresse du navigateur après que l'utilisateur ait appuyé sur le bouton submit.



La requête reçue est ce qui vient après le ? dans l'adresse formée par le navigateur.

Donnees bien recues !

La réponse du serveur s'affiche dans le navigateur.

```
"""
Programme en Python sur le serveur

"""
from socket import socket, AF_INET, SOCK_STREAM
s=socket(AF_INET, SOCK_STREAM)
s.bind(("", 13450))  #  Le serveur est à l'écout
s.listen(5)
test = True
while test :
    connexion, adresse = s.accept()
    print("connexion de : ", adresse)
    req = connexion.recv(1024).decode()
    if req!= "":
        print("La requête reçue : ",req)
        connexion.send(b"Donnees bien recues !")
        if req == "Fin" :
            test = False
connexion.close()
s.close()
```

```
In [3]: runfile('C:/Users/Utilisateur/serveur1.py',
wdir='C:/Users/Utilisateur')
connexion de :  ('127.0.0.1', 55998)
La requête reçue :  GET /?visit_nom=Jacques
+DUBOIS&visit_mail=jdubois%40avesnieres.fr HTTP/1.1
Host: 127.0.0.1:13450
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64; rv:75.0) Gecko/20100101 Firefox/75.0
Accept: text/html,application/xhtml
+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-
US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


connexion de :  ('127.0.0.1', 56001)
La requête reçue :  GET /favicon.ico HTTP/1.1
```

8.4.2. Formulaire avec JavaScript

- La méthode POST est utilisée dans le formulaire donc les données n'apparaitront dans l'adresse formée.
- De plus, le langage JavaScript est utilisé donc il peut y avoir des fonctions telles que la vérification du contenu avant l'envoi du formulaire.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Formulaire avec javascript</title>
    <script langage="JavaScript">

function envoi(){
  if(confirm("confirmation de l'envoi ?")){
    alert("bien envoyé "+formulaire.nom.value);
    return true;
  }
  else{
    return false;
  }
}

function validation(){
  if (formulaire.nom.value == ""){
    alert("champ obligatoire");
    formulaire.nom.focus();
    return false;
  }
}

function nouveau(){
  formulaire.nom.focus();
}
    </script>
</head>

<body>
    <h1>Contact</h1>
    <form name="formulaire" method="post" action="http://127.0.0.1:13450" onSubmit="return envoi()">
        <p>Un formulaire avec Javascript</p>
        <p>
        <label for="nom">Nom</Label>
<input type="text" name="visit_nom" id="nom" autofocus/>
<input type="submit" value="Envoyer" onClick="return validation()">
<input type="reset" value="Annuler" onClick="return nouveau()">
        </p>
    </form>
</body>
</html>
```

Script de vérification du formulaire avant envoi au serveur

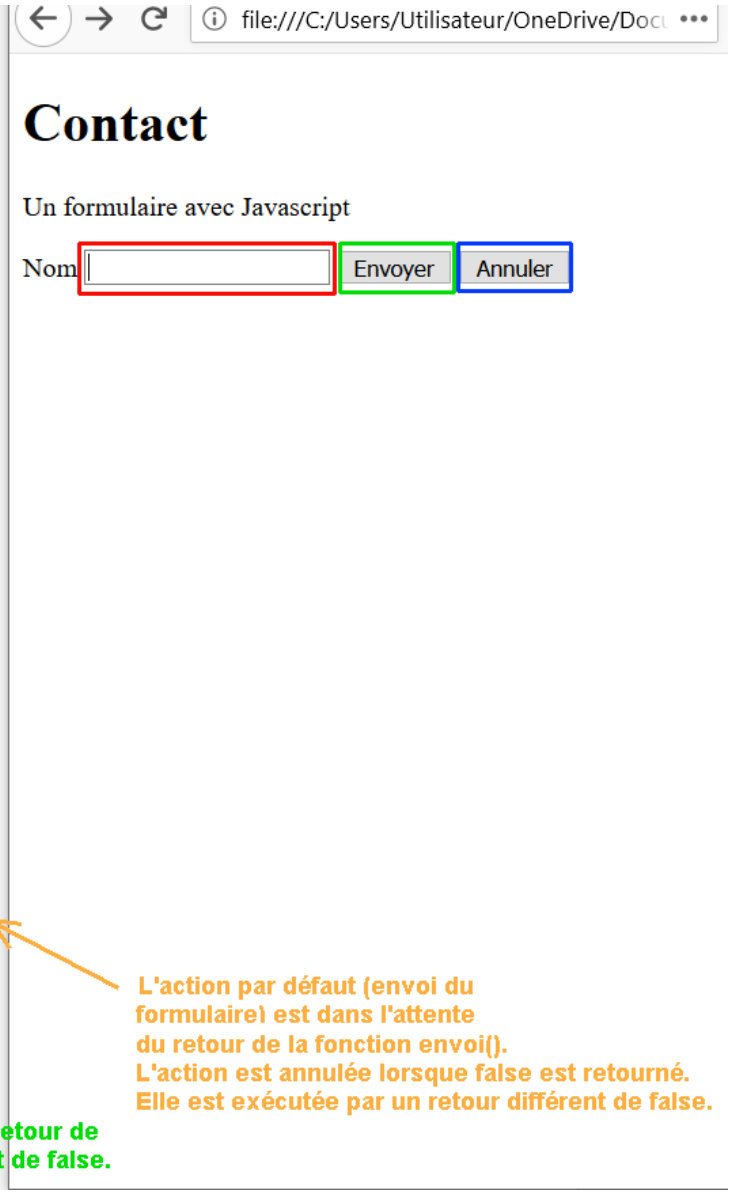
Champ de saisie

Bouton valider

Bouton effacer le formulaire

La fonction nouveau() n'a pas de return.
Elle renvoie toujours undefined qui est différent de false. L'action sera toujours exécutée.

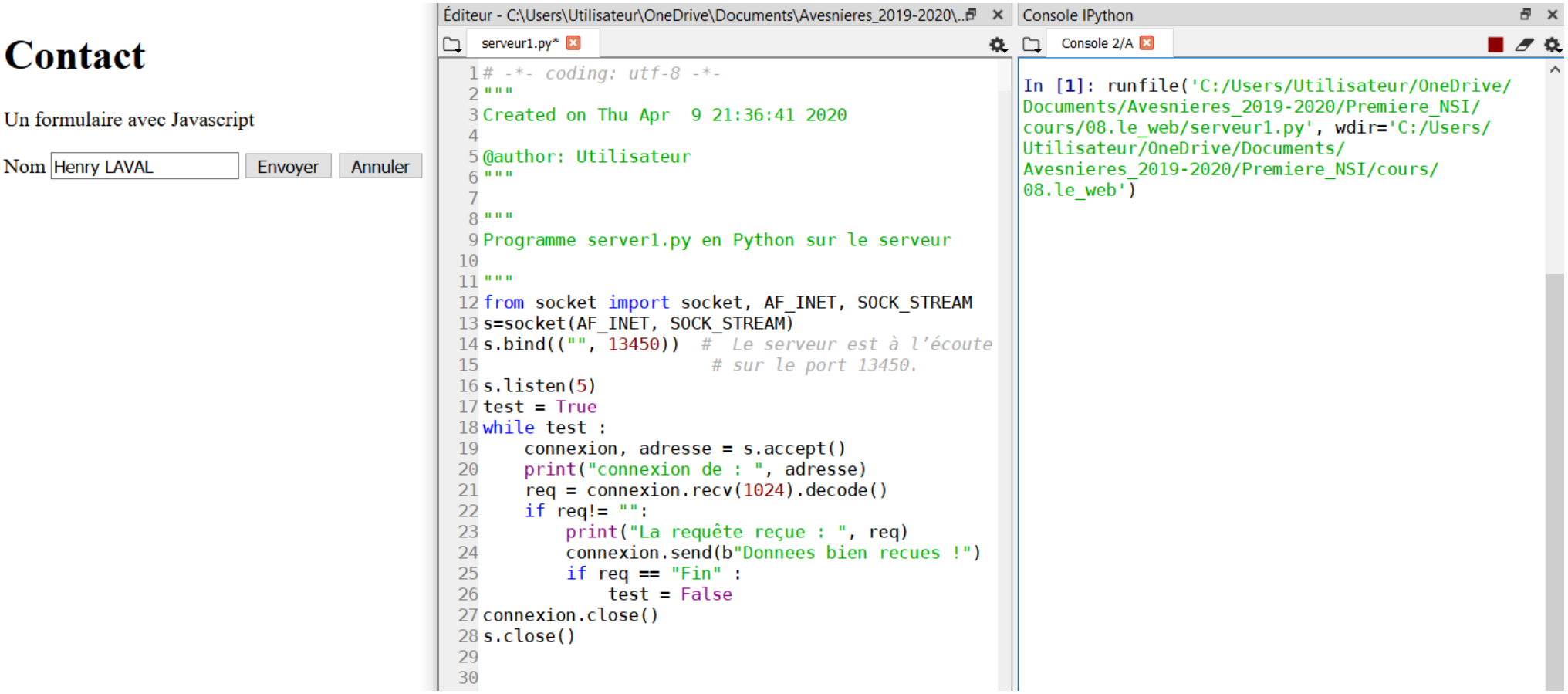
L'action sera exécutée par un retour de la fonction validation() différent de false.



L'action par défaut (envoi du formulaire) est dans l'attente du retour de la fonction envoi().
L'action est annulée lorsque false est retourné.
Elle est exécutée par un retour différent de false.

- **Ce que reçoit le serveur lorsque l'utilisateur a rempli le champ « nom » et appuyé sur le bouton Envoyer :**

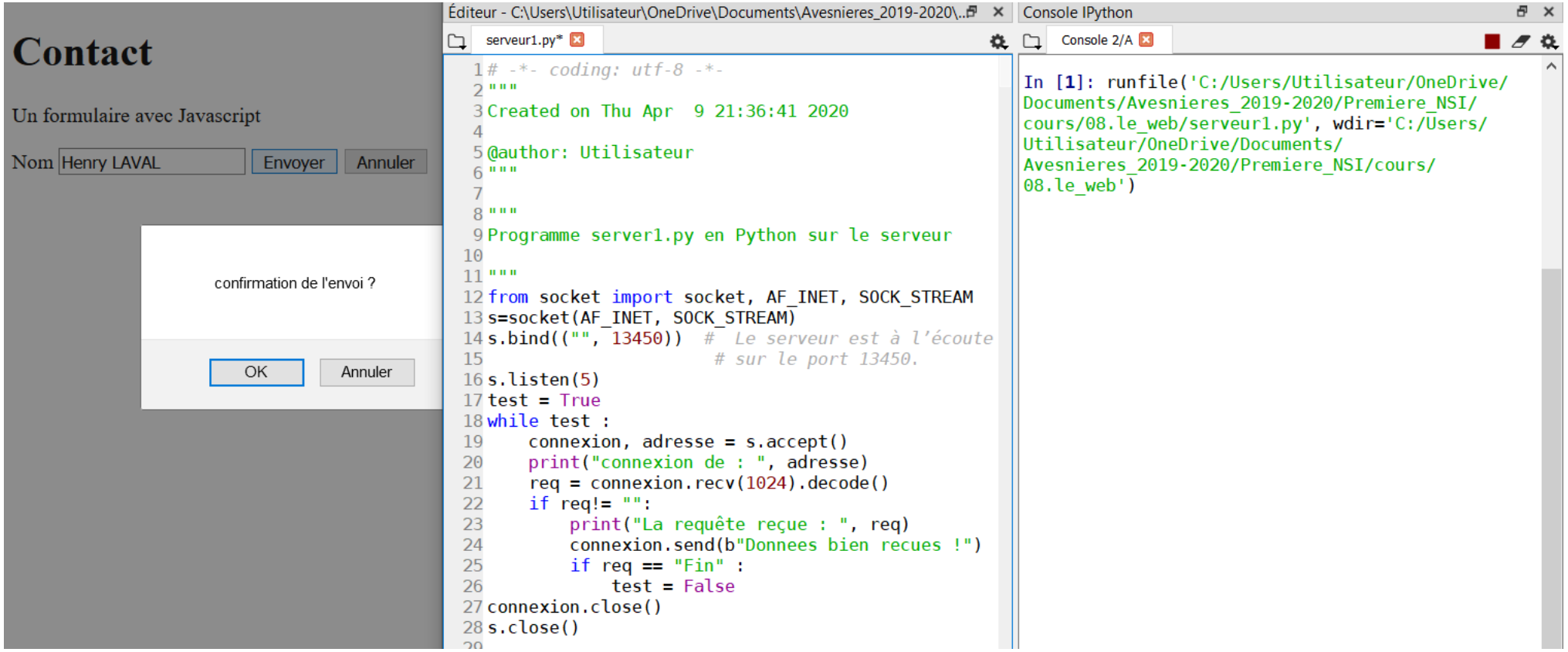
1) Avant d'appuyer sur le bouton Envoyer :



2) Détail de l'enchainement des vérifications JavaScript : Si on clique sur le bouton « Envoyer » alors la fonction validation() est exécutée. Si la valeur de *formulaire.nom.value* n'est pas vide alors la fonction validation retourne la valeur `undefined` qui n'est pas `false` ce qui autorise l'évènement onClick sur le bouton de soumission.

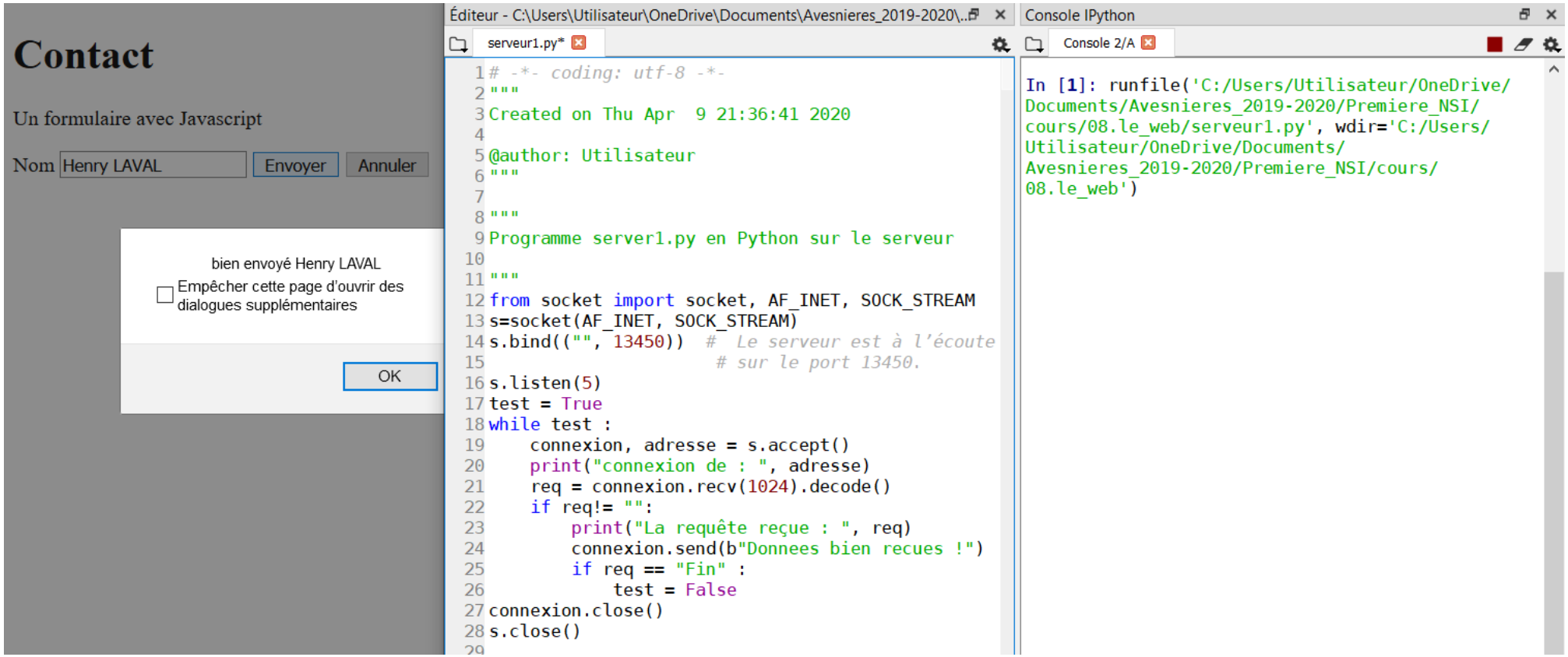
Remarque : Quand nous créons n'importe quel *id* en HTML nous créons une variable portant ce nom. Et on peut accéder à cette variable dans le **JavaScript. Par exemple <input type= "text" name= "visit_nom " id="nom"> . On peut maintenant utiliser la valeur `formulaire.nom.value` dans les fonctions validation() et envoi().**

3) Cela déclenche l'événement onSubmit du formulaire qui exécute alors la fonction envoi(). Celle-ci affiche une boite de confirmation :

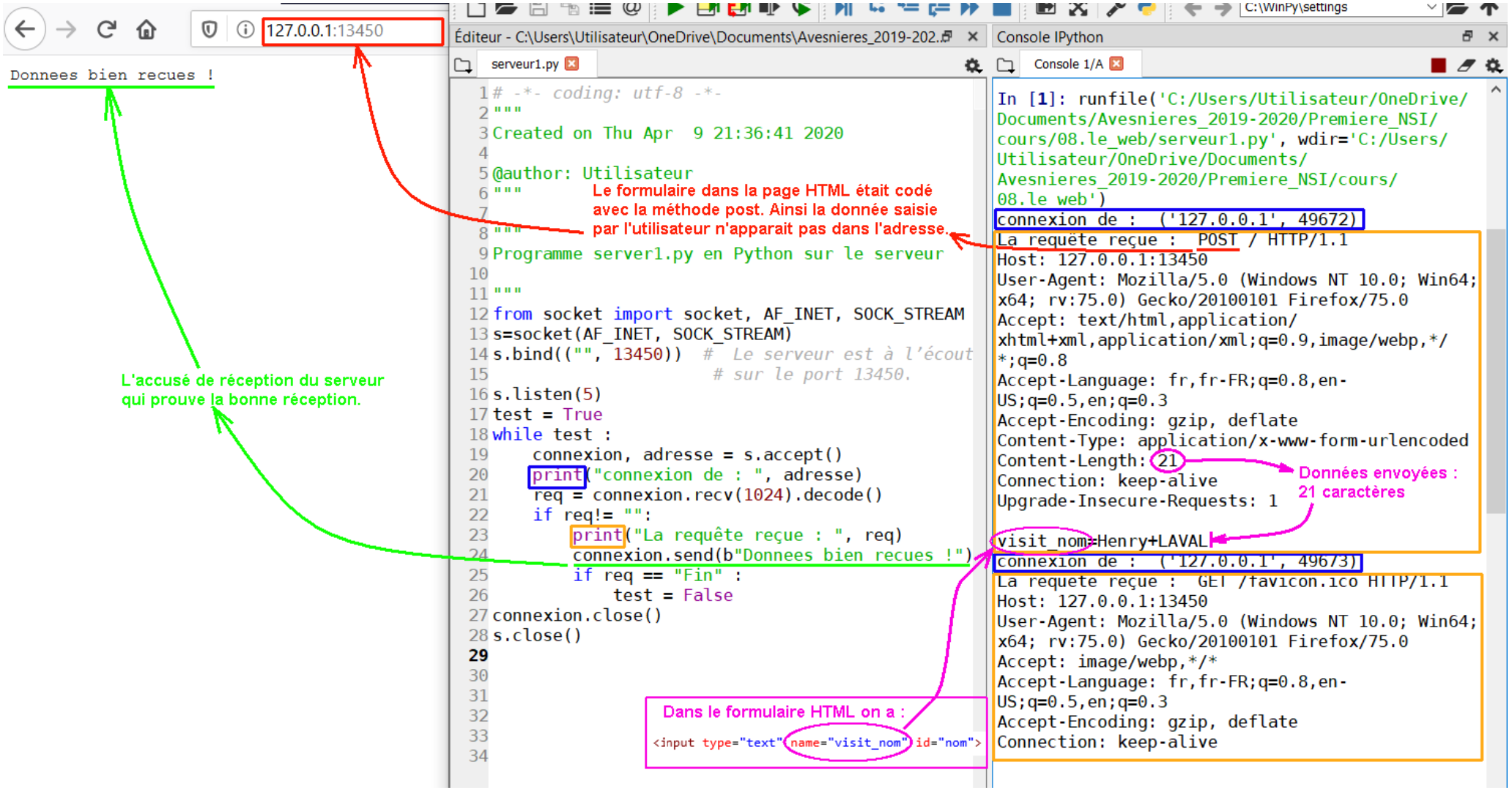


4) Si l'utilisateur clique sur OK dans la boite de confirmation alors l'exécution de la fonction envoi() se poursuit par l'affichage d'une boite d'alerte avec le message « bien envoyé + valeur du champ nom ». L'exécution de la fonction envoi() est suspendue tant que l'utilisateur n'a pas cliqué sur « OK » sur la boite d'alerte.

A ce moment, le serveur n'a encore rien reçu.



5) Si l'utilisateur dans la boite d'alerte clique sur OK alors l'exécution de la fonction envoi() se poursuit et retourne la valeur booléenne `true` et l'évènement onSubmit qui survient alors. Le formulaire est envoyé à ce moment au serveur avec la méthode POST.
6) La requête POST arrive sur le serveur. Le programme Python du serveur accepte la connexion. Il affiche dans la console l'adresse du client qui a envoyé la demande de connexion et la requête.



L'instruction connexion.send(un objet de type byte) permet d'envoyer au navigateur du client un message. On remarque que la donnée entrée dans le formulaire est reçue par le serveur en tant que variable. **Le nom de la variable `visit_nom` est le nom du champ de saisie** <input>.

8.4.3 Formulaire avec php

1. Placer les fichiers **formulaire_page_reponse.html** et **page_rep1.php** sur le serveur, dans un dossier *où du code php peut être exécuté* (par exemple /var/www/html/minisite sur Raspberry ou C:/xampp/htdocs/minisite sur un ordinateur Windows).
2. D'autre part, le serveur php doit être en train d'être exécuté (c'est le cas sur Raspberry si on installé le serveur php avec `sudo apt install php php-mbstring` Sous Windows, si on a installé xampp, il faut l'exécuter en double cliquant sur le fichier C:/xampp/xampp_start.exe).
3. Enfin le fichier **formulaire_page_reponse.html doit être ouvert en saisissant dans la barre d'adresse localhost/minisite/ formulaire_page_reponse.html**

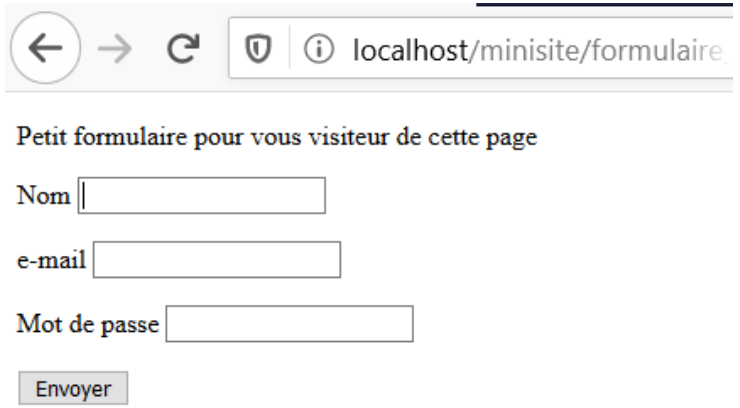
Code HTML du fichier formulaire_page_reponse.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>  Formulaire </title>
        <meta charset="utf-8">
    </head>

    <body>
        <p> Petit formulaire pour vous visiteur de cette page </p>
        <form method="get" action="page_rep1.php">
        <p>
        <label for="nom">Nom</label>
        <input type="text" name="utilisateur" id="nom" autofocus/>
        </p>
        <p>
        <label for="mail">e-mail</label>
        <input type="email" name="adresse" id="mail"/>
        </p>
        <p>
        <label for="passe">Mot de passe</label>
            <input type="password" name="mdp" id="passe" required/>
        </p>
        <input type="submit" value="Envoyer"/>
        </form>
    </body>
</html>
```

La page qui s'affichera quand l'utilisateur aura soumis le formulaire

Affichage



Code HTML + php du fichier page_rep1.php *sur le serveur* :

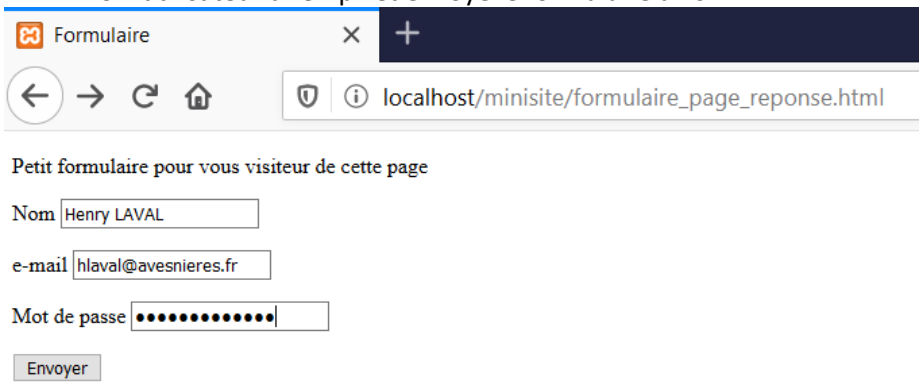
```
<!DOCTYPE html>
<html>
<head>
    <title>  Formulaire </title>
    <meta charset="utf-8">
</head>
<body>
<div>
    <?php $ip=$_SERVER['REMOTE_ADDR'];
    echo 'Votre adresse IP : '.$ip; ?>
    < !—En php, une variable est précédée du caractère $ -->
    <h2>Réponses du serveur</h2>
    <?php if (isset($_GET['utilisateur']) AND isset($_GET['adresse'])
    AND isset($_GET['mdp'])) // Si les variables existent
    {
    $rep1=$_GET['utilisateur']; $rep2=$_GET['adresse']; $rep3=$_GET['mdp'];?>

    <p><?php echo $rep1.' votre email est '. $rep2;?></p>
    <p><?php echo 'et votre mot de passe est '. $rep3;?></p>
    <?php }
    else {echo 'il y a un problème !';?>
        <script langage= "JavaScript">
        alert("il y a un problème ?");
        </script>
    <?php }?>
</div>
</body>
</html>
```

Balises de début et de fin de code php

Il y a 6 sections de code php qui sont exécutées sur le serveur, mais qui n'apparaitront pas dans le code HTML renvoyé par le serveur.
A la place, il y a éventuellement le code HTML produit par le code php.

- Si l'utilisateur a rempli et envoyé le formulaire ainsi :



- **Code HTML du fichier page_rep1.php** envoyé *par le serveur* sur le *navigateur*

```
<!DOCTYPE html>
<html>
<head>
    <title>  Formulaire </title>
    <meta charset="utf-8">
</head>

<body>
<div>
    Votre adresse IP : ::1
    <!-- une variable est précédée du caractère $ -->
    <h2>Réponses du serveur</h2>


    <p>Henry LAVAL votre email est hlaval@avesnieres.fr</p>
    <p>et votre mot de passe est MonMotDePasse</p>
    </div>
</body>
</html>
```

- Affichage sur le navigateur :

```php
<!DOCTYPE html>
<html>
<head>
    <title>  Formulaire </title>
    <meta charset="utf-8">
</head>
<body>
<div>
    <?php $ip=$_SERVER['REMOTE_ADDR'];
    echo 'Votre adresse IP : '.$ip; ?>
    < !–En php, une variable est précédée du caractère $ -->
    <h2>Réponses du serveur</h2>
    <?php if (isset($_GET['utilisateur']) AND isset($_GET['adresse'])
    AND isset($_GET['mdp'])) // Si les variables existent
    {
    $rep1=$_GET['utilisateur']; $rep2=$_GET['adresse']; $rep3=$_GET['mdp']; ?>

    <p><?php echo $rep1.' votre email est '. $rep2;?></p>
    <p><?php echo 'et votre mot de passe est '. $rep3;?></p>
    <?php }
    else {echo 'il y a un problème !';?>
        <script langage= "JavaScript">
        alert("il y a un problème ?");
        </script>
    <?php }?>
</div>
</body>
</html>
```

Balises de début et de fin de code php

Il y a 6 sections de code php qui sont exécutées sur le serveur, mais qui n'apparaitront pas dans le code HTML renvoyé par le serveur.
A la place, il y a éventuellement le code HTML produit par le code php.

- Les fichiers PHP peuvent contenir un mélange de code HTML et PHP. Les sections de code PHP doivent être incluses entre les balises <?php et ?>.
- Lors de l'exécution du fichier PHP par le serveur, le code HTML est recopié inchangé et le code PHP est interprété et remplacé par le résultat.
- Lorsque PHP répond à un formulaire, il récupère les variables du formulaire *via* le tableau associatif $_GET indexé par les noms des champs du formulaire HTML. (On pourrait à la place utiliser le tableau $_POST).

- Si une valeur n'existe pas, à cause par exemple d'une erreur de code html dans le fichier formulaire_page_reponse.html :

<input type="email" name="addresse" id="mail"/> au lieu de
<input type="email" name="adresse" id="mail"/>

- **Code HTML du fichier page_rep1.php** envoyé *par le serveur* sur le *navigateur*

```html
<!DOCTYPE html>
<html>
<head>
    <title>  Formulaire </title>
    <meta charset="utf-8">
</head>

<body>
<div>
    Votre adresse IP : 127.0.0.1
    <!-- une variable est précédée du caractère $ -->
    <h2>Réponses du serveur</h2>

    il y a un problème !        <script langage= "JavaScript">
        alert("il y a un problème ?");
        </script>
    </div>
</body>
</html>
```

- Affichage sur le navigateur :