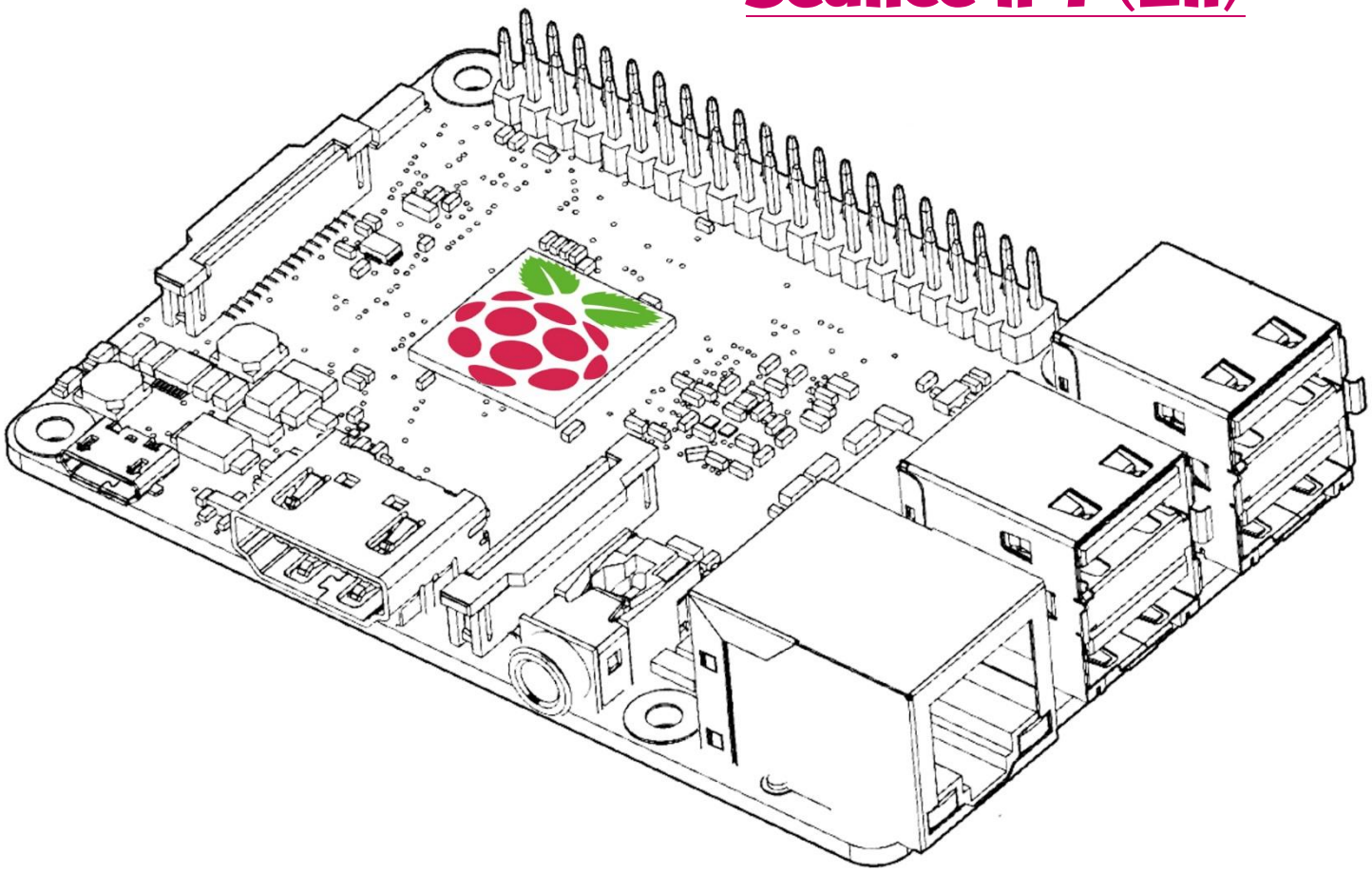
AT31 - Module IPE

IUT RENNES

Séance n°4 (2h)



Serveur web : découverte de Flask

Michaël BOTTIN

Objectifs de la séance :

- Installer un serveur web sur la Raspberry
- Créer ses propres pages web pour contrôler la carte 'Sense Hat'

Récupération des fichiers utiles pour cette séance :

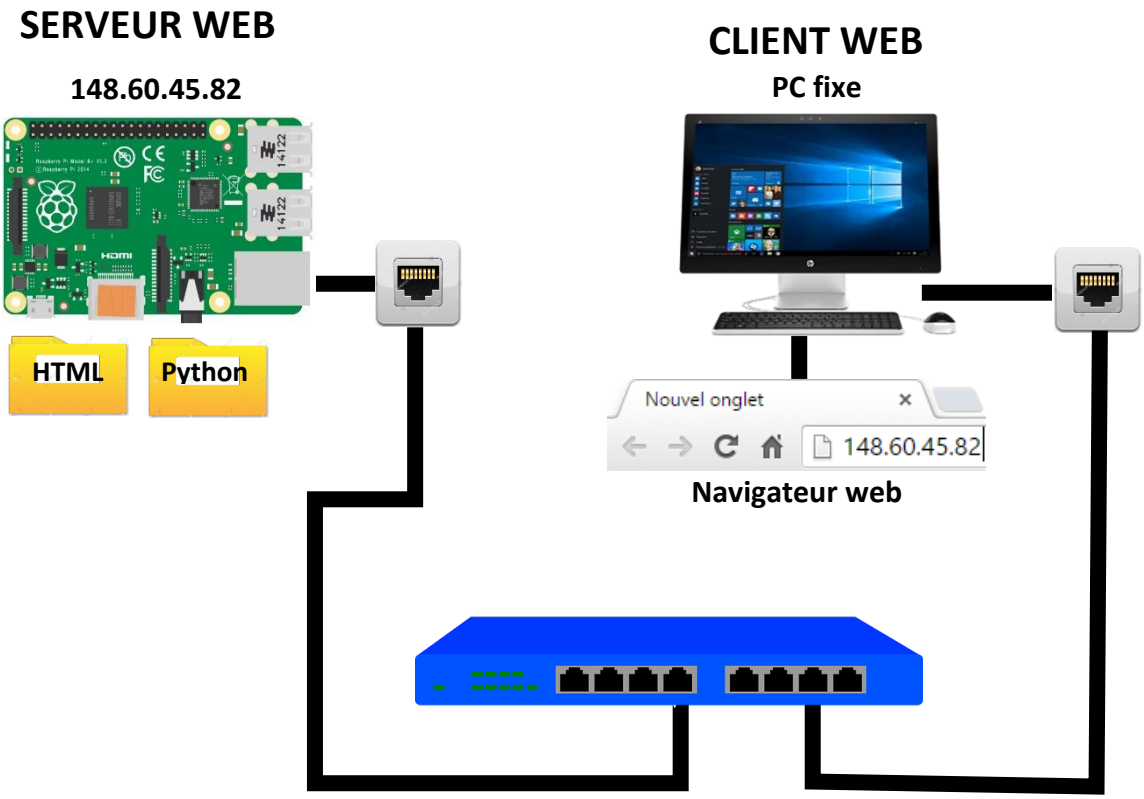
Pour cela, tapez la commande suivante dans le terminal :
$ git clone https://github.com/MbottinIUT/IUT_IPE_TP4.git
[rem : il n'y a pas d'espace dans ce lien, juste des '_']

Une fois la tâche effectuée, vous devriez retrouver, en ouvrant un explorateur de fichiers, un dossier '**IUT_IPE_TP4**' sous '**/home/pi**'.

TUTO 1 : 'Tuto01.py'

Pour les prochains exercices, on va travailler sur le réseau de manière différente que dans les 6 premiers exercices. On va utiliser l'autre sens de communication. Ici, on va chercher à agir sur la Raspberry à distance mais à travers une page web que l'on va nous-même créer. Dans un premier temps, on va se contenter d'afficher un message fixe sur la page s'affichant dans un navigateur.

Pour cela, il faut une architecture de type 'Client-Serveur' :



Dans cet exemple, un serveur web tourne sur la Raspberry du poste n°2 (IP : 148.60.45.82) en attendant les requêtes issues du navigateur web du client (PC du même poste).

Le serveur doit donc embarquer la page HTML qui sera affichée par défaut lorsque le client se connectera sur l'adresse du serveur. Dans notre cas, la page web ne se contentera pas d'afficher du contenu HTML fixe, mais on y intégrera des informations provenant des capteurs de la carte '**Sense Hat**' notamment.

Avant toute chose, il nous faut donc un serveur qui tourne sur notre Raspberry pour qu'elle puisse être interrogée à distance via un navigateur web quelconque.
De nombreuses solutions sont disponibles, au point que l'on pourrait vite s'y perdre. Ces solutions nécessitent souvent de connaître d'autres langages permettant de rendre les pages web dynamiques. On va ainsi retrouver du PHP, du Javascript (JS), du Perl, du Ruby, l'utilisation de Websockets, … Mais dans le cadre de ces TPs, on va rester sur l'utilisation du Python.
Il y aura tout de même un peu de code HTML pour la mise en forme de la page.

Maintenant que l'on a défini le langage que l'on va utiliser pour gérer nos pages web sur le serveur, il faut choisir quel '**framework**' utiliser car là aussi, les solutions Python sont multiples : Django, Pyramid, Web2py, TurboGears, Bottle, Flask, CherryPi, Tornado, Dash, Falcon, Cubicweb, …

Le choix de la solution dépasse le cadre de ce module. Pour notre usage, on pourrait utiliser n'importe lequel de la liste précédente. Mais, quand on a le choix, il vaut mieux utiliser une solution très populaire de manière à trouver facilement de la documentation.
Les deux solutions les plus populaires sont '**Django**' et '**Flask**'. On va ici utiliser '**Flask**' car il est plus facile d'accès. La page officielle est la suivante :

http://flask.pocoo.org/

La première étape consiste donc à télécharger et installer un serveur web.
'**Flask**' est installé en natif avec l'OS Raspbian Buster. Toutefois, mieux vaut le vérifier et le cas échéant le mettre à jour :
$ sudo pip3 install flask

A partir de là, on va pouvoir écrire notre programme Python et notre page HTML. Commencez par écrire dans '**Thonny**' le code suivant avec le nom de fichier '**Tuto01.py**' :

```
# SERVEUR WEB - AFFICHAGE FIXE

# Importation des modules nécessaires
from flask import Flask

# Instanciation du serveur
serveur_web = Flask(__name__)

# Définit le texte à afficher selon le chemin de l'URL
@serveur_web.route('/')      # page racine
def racine() :
    return "Bonjour, vous êtes bien connecté à la Raspberry !!"

@serveur_web.route('/test/')     # page test
def test() :
    return "Changement de page... En construction"

# PROGRAMME PRINCIPAL
serveur_web.debug = False
serveur_web.run(host="0.0.0.0")
```

Comme il s'agit d'un TP de découverte autour de l'aspect serveur du Raspberry, on ne va donc pas rentrer dans le détail du code, mais quelques points nécessitent tout de même d'être précisés :

- '**@serveur_web.route('chemin_URL')**' : le code qui suit sera exécuté lorsque dans le navigateur client, on se connectera à l'adresse 'adresse_IP_serveur/chemin_URL'. Cela peut être du code python. Pour le moment, on ne renvoie juste que l'affichage d'un texte fixe.
- '**serveur_web.debug**' : False ou True, cet attribut permet ou non de valider l'affichage des messages d'erreur dans le navigateur quand une erreur se produit.
- '**serveur_web.run(host= « 0.0.0.0 »)**' : Cette commande permet de mettre en route le serveur web. Tant que cette commande n'est pas exécutée, aucune requête distante depuis un navigateur client ne pourra aboutir. L'adresse « 0.0.0.0 » permet de spécifier que ce serveur sera visible par tous sur le réseau sur lequel il est installé.

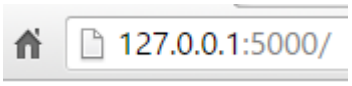
Toutefois, peut-être avez-vous vu en réseau qu'une adresse IP ne suffit pas, il faut également un port (entendez par port un canal de communication qui permet aux données reçues ou émises de savoir par quel service/programme elles doivent être traitées). Ainsi, par exemple, le port 21 est dédié au FTP, le port 80 à l'HTTP, le port 25 au SMTP, …
Le port par défaut pour le serveur Flask est '**5000**'. On peut bien sûr le modifier en ajoutant comme paramètre de méthode '**port='xxxx'**' avec xxxx le numéro désiré (de 0 à 65535). Attention toutefois à ne pas utiliser le même canal que d'autres services. Dans notre cas, nous resterons sur le port par défaut, à savoir '5000'. Il n'est donc pas nécessaire de le préciser au sein de la méthode '**run()**'.

Lancer votre programme depuis '**Thonny**', vous devriez voir apparaître dans le terminal le message suivant :

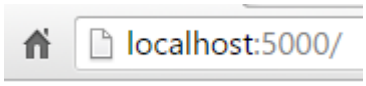
```
>>> %Run Tuto01.py
 * Serving Flask app "Tuto01" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Celui-ci indique que le serveur est en route…

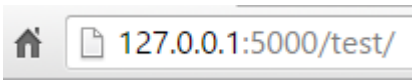
On va commencer par contrôler le bon fonctionnement de notre page en local, c'est-à-dire que l'on va utiliser le navigateur web de la Raspberry (chromium) pour interroger le serveur tournant sur la Raspberry.
Ouvrez donc le navigateur côté Raspberry et testez les adresses suivantes :



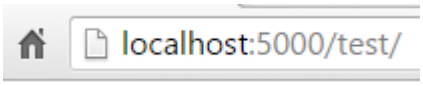
ou



➔ Affiche la page renvoyée par la fonction '**racine()**' en mode local



ou



➔ Affiche la page renvoyée par la fonction '**test()**' en mode local

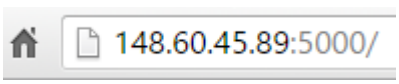
Ce mode local interroge depuis le navigateur client Raspberry, le serveur lancé également sur votre Raspberry et permet donc de vérifier le bon fonctionnement de votre programme sans passer par le réseau pendant son développement.
Dans la zone '**terminal**' de '**Thonny**', vous devriez notamment voir les différentes requêtes (i.e. chargements de pages) effectuées auprès de ce serveur durant son fonctionnement :

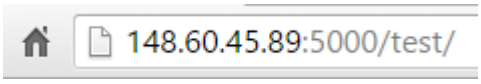
```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Aug/2019 15:35:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2019 15:35:55] "GET /test/ HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2019 15:37:26] "GET / HTTP/1.1" 200 -
```

La 1ère et la 3ème requêtes 'GET' concernent le chargement de la page 'racine' tandis que la 2ème concerne le chargement de la page 'test'.

Toutefois, le but final étant généralement d'interroger le serveur web de la Raspberry à distance, il convient de basculer maintenant vers le PC de votre poste de travail. Ouvrez-y également un navigateur mais cette fois-ci, remplacez l'adresse '127.0.0.1' par l'adresse de votre Raspberry Pi.

Exemple avec la Raspberry du poste n°9 :



puis


Remarque : il est évident que le serveur web de la Raspberry distante doit avoir été lancé pour que votre requête aboutisse, autrement dit que le programme '**Tuto01.py**' soit en cours d'exécution sur la Raspberry interrogée.

TUTO 2 : 'Tuto02.py'

Dans le tutoriel précédent, on ne retournait que du texte qui était affiché sur une page web par défaut.

On va chercher à enrichir simplement cette page web en remplaçant la page par défaut par la nôtre.

On va donc :

- Modifier la taille et le style de la police d'affichage
- Ajouter une image sur chaque page web
- Mettre une icône sur l'onglet de la page

Bien sûr, le but ici n'est pas de vous enseigner les langages du web HTML & Co, mais de travailler sur la Raspberry. Toutefois, pour mieux comprendre ce que vous allez écrire, mieux vaut quelques bases…

➔ Le code HTML n'est pas un langage de programmation, mais un langage de description qui fonctionne avec des balises que l'on ouvre (exemple : <body>) et que l'on ferme (exemple : </body>)

➔ Ces balises répondent à des mots-clefs clairement définis dans le langage. En voici quelques exemples :

- **<head>** : entête
- **<title>** : titre
- **<body>** : corps de la page
- **<h1>** : texte au format 1
- **<h5>** : texte au format 5
- **<b>** : gras
- **<i>** : italique
- **<u>** : souligné
- **
** : retour chariot
- **<font>** : police du texte
- **<form>** : formulaire
- (…)

➔ Elles peuvent s'imbriquer les unes dans les autres

➔ Et pour rendre une page web dynamique, on peut y intégrer des variables en les entourant de double accolades (sections) : **{{variable}}**. Ceci se fait grâce à une compatibilité de '**Flask**' avec '**Jinja**'.

➔ Le texte entre les balises **<link>** permet notamment l'affichage de l'icône de la page dans l'onglet du navigateur.

➔ La balise suivante permet d'afficher une image : **<img src={{ url_for('static', filename='NomFichier.jpg') }}>**

On va maintenant écrire le code HTML de nos deux pages web. N'importe quel éditeur de texte peut convenir, mais un éditeur multi-langage est disponible sur l'OS de la Raspberry. On va donc en profiter.
Via le menu '**Programmation**', ouvrez l'éditeur '**Geany**' :

Créez un premier fichier '**Tuto02_page01.html**' et mettez-y le code suivant :

```
<html>
        <head>
            <title>Tuto02</title>
            <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
        </head>
        <body>
            <h3>Bonjour, vous êtes bien connectée à votre Raspberry !!</h3>
            <img src={{ url_for('static', filename='raspberrypi.jpg') }}>
        </body>
</html>
```

Pour expliquer brièvement le contenu de ce fichier, on y trouve :

- Les balises <html> et </html> entre lesquelles on retrouve l'ensemble du code HTML de notre page.
- Les balises <head> et </head> entres lesquelles on va retrouver les infos à afficher dans la fenêtre du navigateur :
  - Le titre de l'onglet entre <title> et </title>
  - L'icône à afficher sur l'onglet dans la balise <link>
- Les balises <body> et </body> contiennent elles ce qui sera réellement affiché sur la page :
  - Du texte avec les balises <h3> et </h3> qui fixent la taille du texte
  - Une image avec la balise <img> où l'on donne le dossier et le nom du fichier de l'image

Créez ensuite un second fichier '**Tuto02_page02.html**' et copiez-y le code suivant :

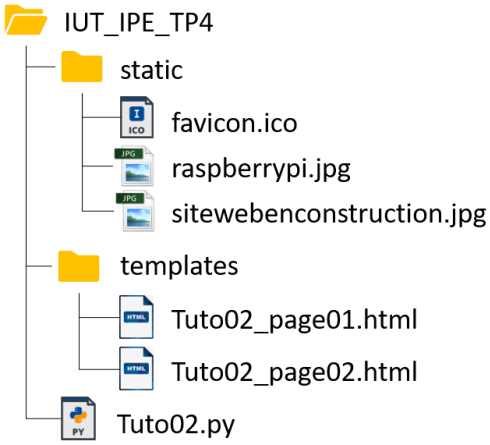
```
<html>
        <head>
            <title>Tuto02</title>
            <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
        </head>
        <body>
            <h3>Changement de page... En construction</h3>
            <img src={{ url_for('static', filename='sitewebenconstruction.jpg') }}>
        </body>
</html>
```

Créez enfin un code python (dans '**Thonny**' cette fois-ci) '**Tuto02.py**' et écrivez le code suivant :

```
# SERVEUR WEB - AFFICHAGE FIXE

# Importation des modules nécessaires
from flask import Flask
from flask import render_template

# Instanciation du serveur
serveur_web = Flask(__name__)

# Définit le texte à afficher selon le chemin de l'URL
@serveur_web.route('/')      # page racine
def racine() :
    return render_template('Tuto02_page01.html')

@serveur_web.route('/test/')     # page test
def test() :
    return render_template('Tuto02_page02.html')

# PROGRAMME PRINCIPAL
serveur_web.debug = False
serveur_web.run(host="0.0.0.0")
```

Par rapport au fichier '**Tuto01.py**', la seule différence est que dans chaque fonction dédiée à une page web, on ne retourne pas un simple texte, mais l'on demande à '**Flask**' d'effectuer l'envoi de la page web dont on spécifie le nom de fichier.
A noter que l'on a également importé un module supplémentaire permettant le rendu de pages web.

ATTENTION !! Pour que tout fonctionne correctement, vos fichiers doivent être sauvegardés en respectant parfaitement l'arborescence suivante :



Le dossier '**static**' contient toutes les ressources de vos pages (ici icône et images).
Le dossier '**templates**' contient tous les fichiers HTML de vos pages
Enfin, le fichier python lançant le serveur doit se situer au même niveau d'arborescence que les dossiers '**static**' et '**templates**'.

Pour le tester, en suivant les instructions du tutoriel précédent, vous pouvez au choix soit le tester en local (avec le navigateur de votre raspberry) ou à distance (avec le navigateur de votre PC). Ce choix sera le vôtre pour tous les prochains exercices également.

Et là aussi, dans la partie '**terminal**' de '**Thonny**', on peu voir les différentes requêtes effectuées :

```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Aug/2019 15:59:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2019 15:59:16] "GET /static/raspberrypi.jpg HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2019 15:59:17] "GET /static/favicon.ico HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2019 15:59:38] "GET /test HTTP/1.1" 301 -
127.0.0.1 - - [23/Aug/2019 15:59:38] "GET /test/ HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2019 15:59:41] "GET /static/sitewebenconstruction.jpg HTTP/1.1" 200
```

TUTO 3 : 'Tuto03.py'

Pour le moment, on a donc vu comment afficher du texte fixe et des images dans une page web distante. Cela présente toutefois un intérêt très limité. Non seulement, il serait intéressant d'afficher autre chose que du texte, mais également de rendre la page dynamique.

Dans ce petit tutoriel, on va donc voir comment intégrer dans notre code HTML des variables qui se mettent à jour grâce à notre code Python.
Ecrivez le code suivant dans '**Thonny**' avec le nom de fichier '**Tuto03.py**' :

```
# SERVEUR WEB - AFFICHAGE FIXE AVEC VARIABLES

# Importation des modules nécessaires
from flask import Flask          # bibliothèque pour serveur web
from flask import render_template   # bibliothèque pour le rendu de page web
import socket
import datetime                     # bibliothèque date/heure

# Instanciation du serveur
serveur_web = Flask(__name__)

# Définit le texte à afficher selon le chemin de l'URL
@serveur_web.route('/')      # page racine
def racine() :
    maintenant = datetime.datetime.now()
    chaine_heure = maintenant.strftime("%H:%M")
    return render_template('Tuto03.html', poste =socket.gethostname(), heure =chaine_heure)

# PROGRAMME PRINCIPAL
serveur_web.debug = False
serveur_web.run(host="0.0.0.0")
```

Là aussi, quelques remarques s'imposent pour aider à la compréhension :

- ➔ '**datetime.datetime.now()**' : permet de récupérer l'heure système.
- ➔ '**render_template**' : cette méthode permet de renvoyer dans le navigateur, non pas un simple texte fixe comme précédemment, mais la page web passée en argument (ici 'Tuto03.html'). Les autres paramètres (**poste** et **heure**) sont des variables dynamiques de notre page web passées également en argument.

Bien sûr, l'écriture du code HTML vous aidera à mieux comprendre. Dans '**Geany**' créez un nouveau fichier que vous nommerez '**Tuto03.html**' :

```
<html>
        <head>
            <title>Test</title>
            <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}"
        </head>
        <body>
            <h2>Vous etes sur le poste {{poste}}</h2>
            <h1>Et il est actuellement {{heure}}</h1>
        </body>
</html>
```

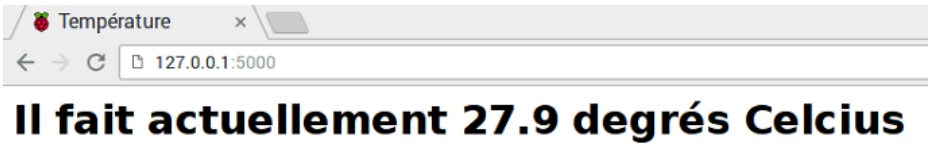
Comme précédemment, le code python est à sauvegarder dans le dossier du TP4 tandis que le code HTML est à sauvegarder dans le dossier '**templates**'.

Quelques explications :
Vous pouvez remarquer les doubles accolades qui permettent de passer une variable dans le code HTML. Bien évidemment, il faut que les noms de variables du code HTML et du code Python soient identiques dans leur forme.

Faire contrôler votre programme par un enseignant.

EXERCICE 1 : 'Ex01.py'

Dans cet exercice, vous allez réaliser de façon plus autonome une page web permettant d'afficher la valeur de la température récupérée sur la carte '**Sense Hat**'. L'interface doit ressembler à ceci :



En vous inspirant de l'exercice précédent et de ce que l'on a fait sur la carte '**Sense Hat**' dans le TP n°3, créer le code python '**Ex01.py**' et la page HTML correspondante '**Ex01.html**' permettant de réaliser ce qui vous est demandé.

Faîtes vérifier le bon fonctionnement de votre interface par un enseignant.

Remarque : Cette température ne se met à jour que si vous demandez un rafraîchissement de la page ou que vous la rechargez. Une mise à jour dynamique et automatique ferait appel par exemple à du code Javascript ou à l'utilisation de websockets qui sortent du cadre de ces TPs.

EXERCICE 2 : 'Ex02.py'

Nous allons maintenant rendre notre page interactive en y intégrant un bouton '**Allumer**' et un bouton '**Eteindre**'. Comme nous n'avons pas de lampe à disposition sur la carte, nous allons éteindre la matrice ou afficher un cercle blanc qui symbolisera notre lampe.

Voici à quoi va ressembler dans un premier temps notre interface web :

L'utilisation de boutons dans une page web fait appel à ce que l'on appelle des *formulaires* (utilisation des balises **<form>** et **</form>**).

Dans '**Geany'**, créez un nouveau fichier HTML, nommez-le '**Ex02.html**' (à stocker dans le dossier '**templates**') et insérez-y le code suivant :

```
<html>
    <head>
        <title>ECLAIRAGE</title>
        <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
    </head>
    <body>
        <h1>COMMANDE DE LA MATRICE A DISTANCE</h1>
        <form method="post" action="{{url_for('change')}}">
            <input type="submit" name="switch" value="Allumer"/>
            <input type="submit" name="switch" value="Eteindre"/>
        </form>
    </body>
</html>
```

On voit apparaître des paramètres dans la balise du formulaire. Le premier fait référence à une '**méthode**'. Cette méthode est de type '**POST**'.
En fait, dans le protocole HTTP dédié à l'échange d'informations entre un serveur et un client, on émet des requêtes (exemple : recherche Google). Les deux principaux types de requêtes qui existent sont :

- ➔ **GET** : On passe ici des paramètres via l'URL. La chaîne de requête est ajoutée à l'URL à la suite d'un ' ?'
- ➔ **POST** : On passe ici des paramètres via le corps du message HTTP, soit souvent par le biais d'un formulaire, d'où son utilisation dans notre cas.

Dans le formulaire, on a également deux balises de type '**input**', c'est-à-dire des balises qui amènent une action de l'utilisateur. Ici, le type '**submit**' correspond à un bouton.

Pour résumer, si on agit sur un des deux boutons, un '**POST**' sera effectué via le formulaire, ce qui lancera l'affichage de la page '**URL_de_base/change/**' (d'où le paramètre **action=''{{url_for('change')}}''**) tout en lui fournissant la valeur ('**value**') du bouton qui a été appuyé.

Il faut maintenant s'intéresser à notre code Python. Je rappelle que si l'on appuie sur le bouton '**Eteindre**', on doit éteindre la matrice de la carte '**Sense Hat**' et que si l'on appuie sur le bouton '**Allumer**', on doit afficher sur la matrice un cercle blanc qui symbolise une lampe.

Créez alors dans '**Thonny**' un nouveau fichier python que vous nommez '**Ex02.py**' et tapez-y le code suivant :

```
# SERVEUR WEB - GESTION ON-OFF DE LA MATRICE

# Importation des modules nécessaires
from flask import Flask          # bibliothèque pour serveur web
from flask import render_template   # bibliothèque pour le rendu de page web
from flask import request, url_for  # bibliothèque pour la gestion dynamique des pages
from sense_hat import SenseHat

# Instanciation de l'objet SenseHat
sense = SenseHat()
b = (255, 255, 255)
n = (0,0,0)

# Tableau représentant la lampe
lampe = [n,n,b,b,b,b,n,n,
         n,b,b,b,b,b,b,n,
         b,b,b,b,b,b,b,b,
         b,b,b,b,b,b,b,b,
         b,b,b,b,b,b,b,b,
         b,b,b,b,b,b,b,b,
         n,b,b,b,b,b,b,n,
         n,n,b,b,b,b,n,n]

# Creation de l'objet serveur base sur Flask
serveur_web = Flask(__name__)

# Definit ce que le serveur renvoie suivant le chemin de l'URL
@serveur_web.route('/')
def principal():
    return render_template('Ex02.html')

@serveur_web.route('/change/', methods=['POST'])
def change():
    if request.method == 'POST' :
        if request.form['switch'] == 'Allumer' :
            sense.set_pixels(lampe)     # Affichage du cercle blanc sur la matrice
        elif request.form['switch'] == 'Eteindre' :
            sense.clear()
    return render_template('Ex02.html')

# PROGRAMME PRINCIPAL
serveur_web.debug = False
serveur_web.run(host="0.0.0.0")
```

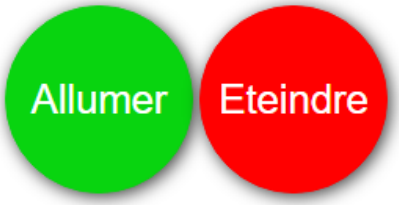
Testez alors le bon fonctionnement de votre programme et **faîtes-le contrôler par un enseignant**.

Remarque : notre interface est sommaire et un peu austère, mais si vous êtes un peu familier avec l'HTML, vous savez peut-être que l'on peut utiliser des feuilles de styles via CSS. Un fichier '**monstyle.css**' a déjà été créé dans le dossier '**static**'. Il contient des classes graphiques '**on**' et '**off**' que j'ai créées pour nos boutons à titre d'exemple.

Pour utiliser ce fichier au sein de notre page, vous devez modifier le code de la page HTML de la manière suivante :

```
<html>
    <head>
        <title>ECLAIRAGE</title>
        <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
        <link rel="stylesheet" href="{{ url_for('static', filename='monstyle.css') }}">
    </head>
    <body>
        <h1>COMMANDE DE LA MATRICE A DISTANCE</h1>
        <form method="post" action="{{url_for('change')}}">
            <input type="submit" name="switch" class="on" value="Allumer"/>
            <input type="submit" name="switch" class="off" value="Eteindre"/>
        </form>
    </body>
</html>
```

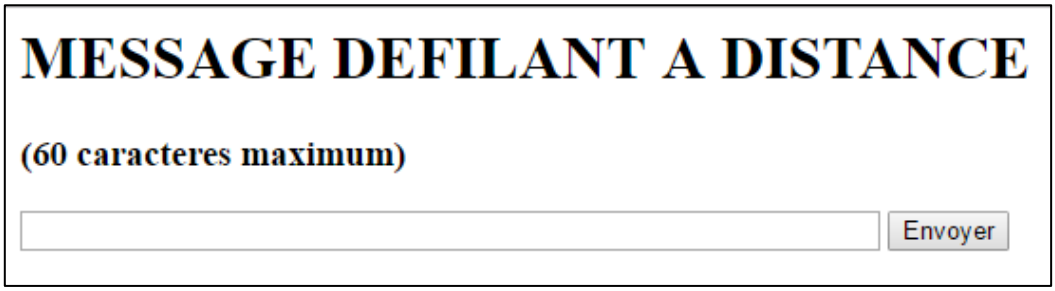
En testant à nouveau notre page web, vous devriez obtenir un résultat plus agréable pour vos yeux :



EXERCICE 3 : 'Ex03.py'

On va enfin, chercher à reproduire ce que l'on a fait dans le TP n°3, à savoir faire défiler un message utilisateur sur la matrice 8x8 de la carte '**Sense Hat**'. Mais maintenant, on va permettre à l'utilisateur de saisir son message à travers une interface web que l'on va créer. Cela pourra donc se faire sur un poste distant du moment qu'il est sur le même réseau.

Cette interface doit disposer d'une zone de saisie de texte et d'un bouton d'envoi :



On va à nouveau commencer par l'écriture du code HTML de la page web sous '**Geany**' dans un fichier que nous nommerez '**Ex03.html**' :

```
<html>
    <head>
        <title>MESSAGE</title>
        <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
    </head>
    <body>
        <h1>MESSAGE DEFILANT A DISTANCE</h1>
        <h3>(60 caracteres maximum)</h3>
        <form method="post" action="{{url_for('change')}}">
            <input type="text" name="message" size="60" maxlength="60"/>
            <input type="submit" name="switch" value="Envoyer"/>
        </form>
    </body>
</html>
```

La nouveauté consiste dans le formulaire à l'insertion d'une zone de texte via la balise '**input**' mais en précisant maintenant le type '**text**'. On en profite pour fixer sa taille à 60 caractères afin de ne pas avoir de trop longs messages.

Dans '**Thonny**', créez un nouveau fichier '**Ex03.py**' et recopiez-y le code ci-dessous en le complétant :

```
# SERVEUR WEB - MESSAGE DEFILANT SUR LA MATRICE

# Importation des modules nécessaires
from flask import Flask         # bibliothèque pour serveur web
from flask import render_template   # bibliothèque pour le rendu de page web
from flask import request, url_for  # bibliothèque pour la gestion dynamique des pages
from sense_hat import SenseHat

# Instanciation de l'objet SenseHat
sense = SenseHat()
bleu = (5, 41, 99)
orange = (252,177,37)

# Creation de l'objet serveur base sur Flask
serveur_web = Flask(__name__)

# Definit ce que le serveur renvoie suivant le chemin de l'URL
@serveur_web.route('/')
def principal():
    return render_template('Ex03.html')

@serveur_web.route('/change/', methods=['POST'])
def change():
    if request.method == 'POST' :
        message_a_afficher = request.form['message']
        # Affichage du message
        ???????????
        # Efface la matrice
        ???????????
    return render_template('Ex03.html')

# PROGRAMME PRINCIPAL
serveur_web.debug = False
serveur_web.run(host="0.0.0.0")
```

Pour ce code Python, vous devez compléter la récupération du '**message**' et son affichage (zones avec ' ????').
Faire contrôler votre programme par un enseignant.

Remarque : Il serait également intéressant de pouvoir sélectionner la couleur du texte et la couleur du fond dans notre interface web. Vous pouvez tester le programme '**Ex03_Color.py**' et jeter un coup d'œil au fichier '**Ex03_Color.html**'.

Intermède :

On va juste faire une petite parenthèse informative avant de terminer cette séance.
Effectivement, comme l'on s'intéresse à la possibilité de communication réseau de la Raspberry, il est important de signaler que les cartes Raspberry sont souvent utilisées en mode '**headless**', c'est-à-dire sans écran, sans clavier et sans souris !!
Comment peut-on alors écrire du code et l'exécuter ? Tout simplement en utilisant le réseau.
On peut donc facilement travailler sur un PC sous Windows et y écrire du code qui sera exécutée sur la Raspberry voisine à condition que celle-ci soit bien sûr ***sur le même réseau.***

Toutefois, il nous faut un outil qui va nous permettre d'utiliser le réseau comme affichage déportée de la console, de la fenêtre d'une application, voire même du bureau entier de la Raspberry vers votre PC.
Là encore, énormément de solutions sont disponibles avec leur lot d'avantages et d'inconvénients.
Je vais vous présenter ici une solution qui est utilisée par le technicien informatique de notre département pour la prise de contrôle d'un PC du parc informatique à distance.
Ce logiciel s'appelle '**MobXterm**'. C'est un logiciel avec un très grand nombre de possibilités. Nous l'utiliserons dans sa version gratuite qui est bien suffisante pour nos besoins.

Je vous donne vraiment ces informations en dehors du cadre d'un exercice car il s'agit d'un outil qui peut être utilisé à tout moment et qui s'avère parfois indispensable suivant la configuration matérielle du système.
En projet également, vous serez peut-être amené à utiliser l'un ou l'autre des deux méthodes que je vais vous présenter.

Installation De MobaXterm sous Windows :

Lorsque vous êtes administrateur de votre poste, vous pouvez installer MobaXterm comme n'importe quel autre logiciel via son fichier 'setup'.
Dans notre cas, le logiciel n'est pas installé sur les PC de la salle. On va donc utiliser sa version portable que vous trouverez sur la page de téléchargement du site officiel :

https://mobaxterm.mobatek.net/download-home-edition.html

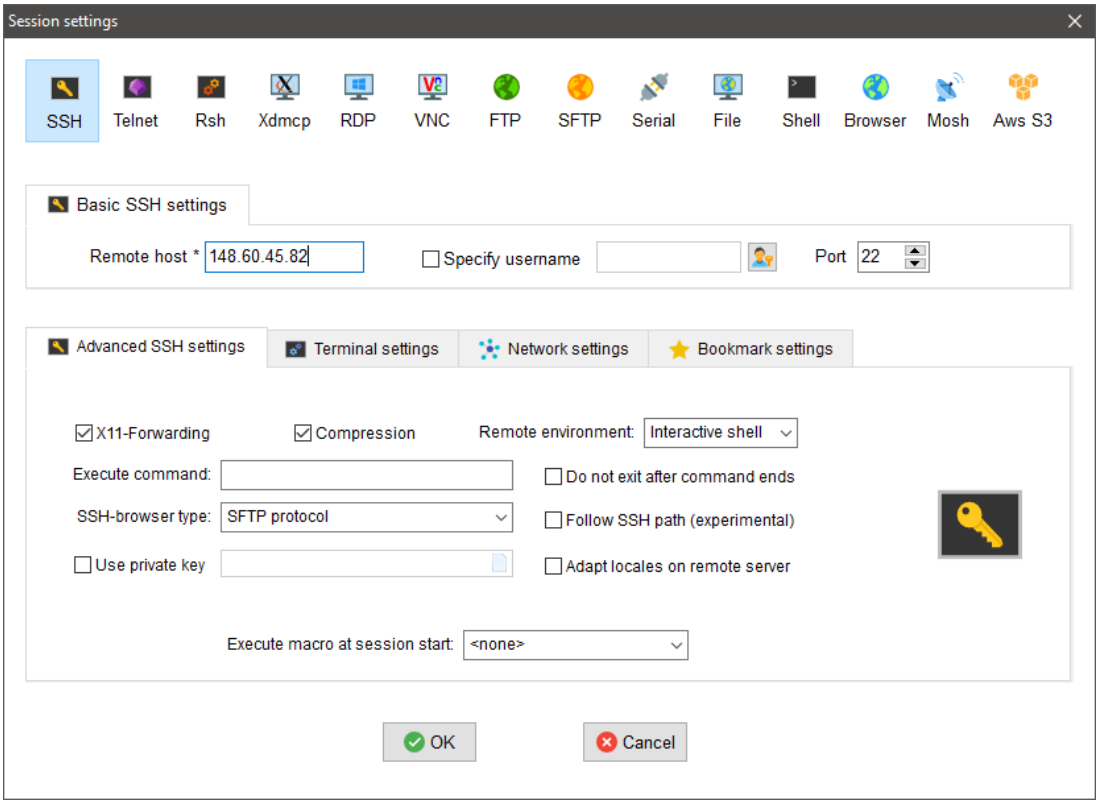
Téléchargez-le, dézippez-le sur le bureau par exemple et lancez le fichier 'exe'.

Utilisation de MobaXterm sous Windows :

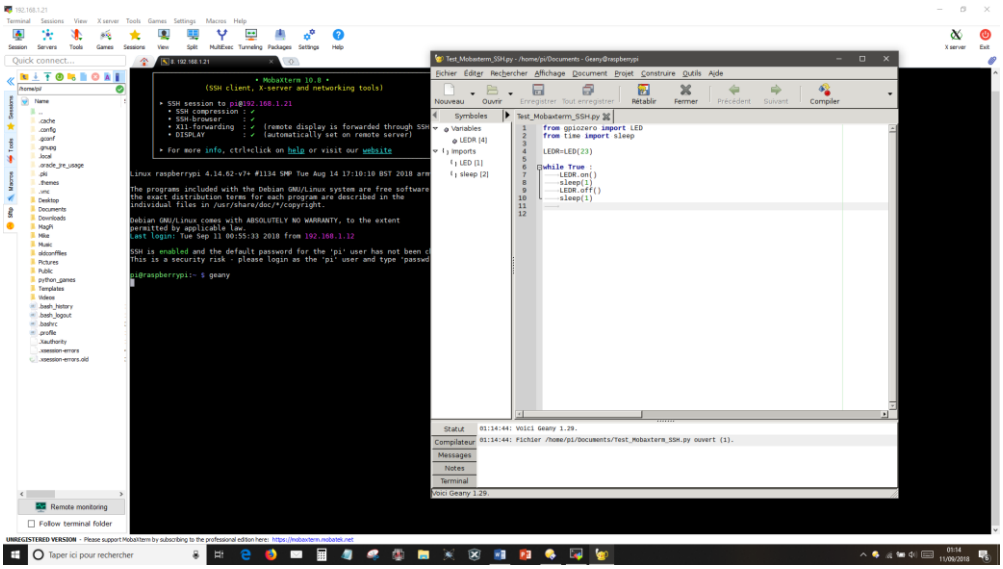
Le premier mode que l'on va découvrir est le '**SSH**', qui permet de récupérer la console/terminal de la Raspberry sous Windows, et de pouvoir également ouvrir la fenêtre d'une application à distance.
Ce protocole de commande sécurisé très courant appelé '**SSH**' (Secure Shell), développé en 1995 pour remplacer plusieurs outils tels que Telnet, FTP, RSH, …
Ce protocole permet à un client (un PC dans notre présentation) de se connecter à une machine distante (serveur : ici la Raspberry) sur le réseau en utilisant une communication chiffrée appelée 'tunnel'.

Pour le mode **SSH** :

- ➔ Valider le 'SSH' dans les options de configuration de la Raspberry (Cf. TP n°1)
- ➔ Côté PC, sous MobaXterm, créer une session en fournissant l'adresse IP de la Raspberry à atteindre :



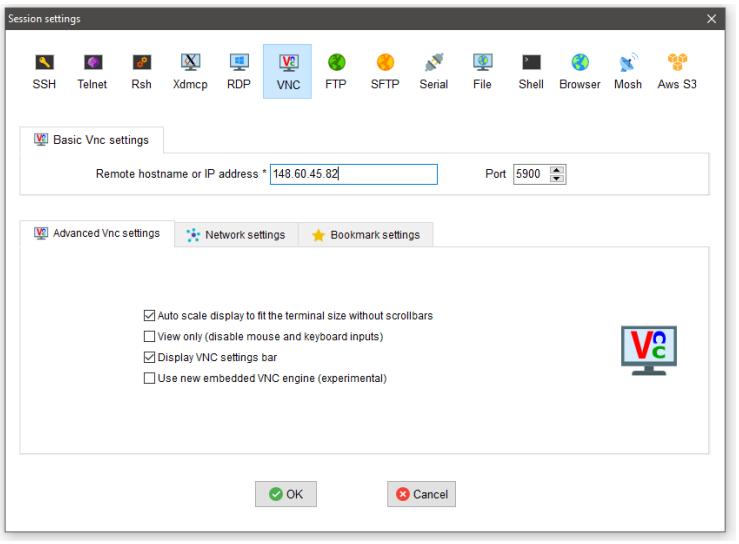
- ➔ Dans cette fenêtre de création de session, vérifier que le 'X11 forwarding' est activé. C'est lui qui permettra l'affichage d'une fenêtre en plus du mode console.
- ➔ Faîtes alors 'OK' pour enregistrer et lancer la session en mode 'SSH'
- ➔ La console va alors s'ouvrir et vous demander un login :
  - Utilisateur : **pi**
  - Mot de passe : **Postexx** (Cf. TP n°1) (s'il n'a pas été changé, c'est 'raspberry')
- ➔ Vous arriverez alors sur prompt de la ligne de commande où vous pouvez taper toutes les commandes habituelles, elles s'exécuteront sur la Raspberry.
- ➔ *Remarque :* avec le '**X11 forwarding**' validé, vous pouvez récupérer la fenêtre d'une application. Si vous tapez par exemple la commande '**thonny**'. La fenêtre de Geany s'ouvrira alors dans Windows (enregistrement, exécution de programme, CTRL+C, … tout fonctionne) (ATTENTION toutefois à ne pas faire 'sudo thonny' car on ne peut lancer des commandes qu'avec les droits de l'utilisateur SSH soit 'pi' ici)



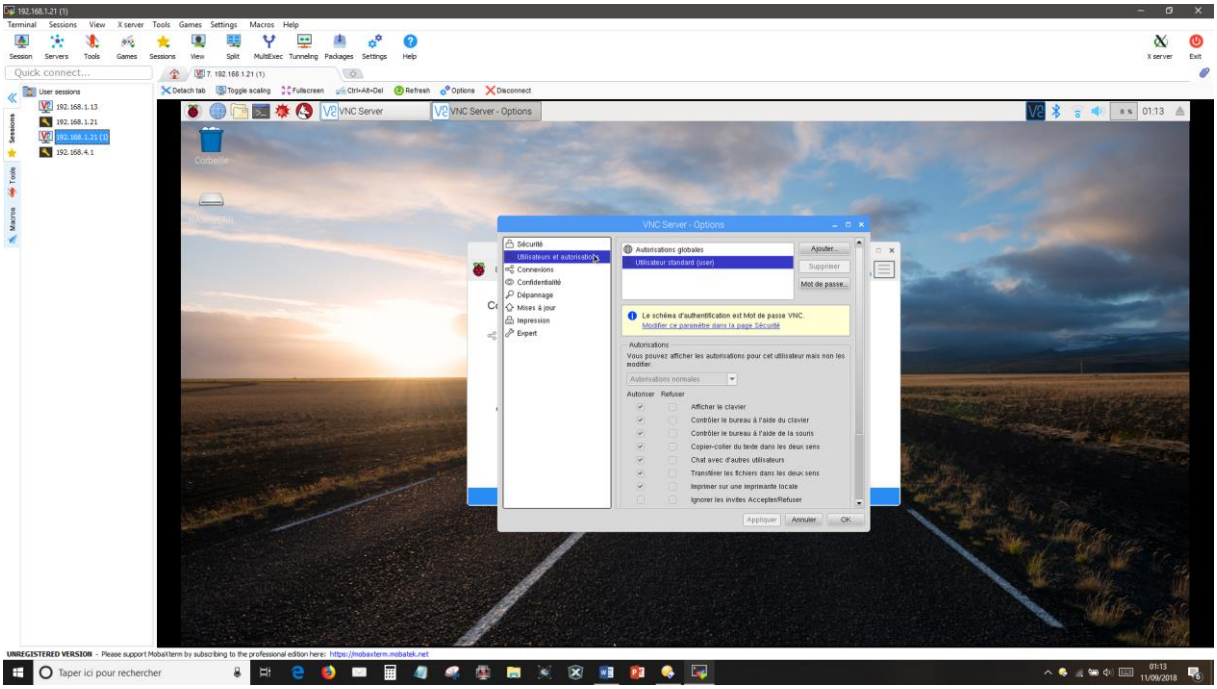
L'autre mode que l'on va décrire ici est le mode '**VNC**' (Virtual Network Computing).
« C'est un système de visualisation et de contrôle de l'environnement de bureau d'un ordinateur distant. Il permet au logiciel client VNC de transmettre les informations de saisie du clavier et de la souris à l'ordinateur distant, possédant un logiciel serveur VNC à travers un réseau informatique » (Source Wikipedia).

<u>En mode VNC :</u>

- ➔ Côté Raspberry, il faut également que cette option (VNC) soit activée dans la configuration de la raspberry (Cf. TP n°1)
- ➔ Côté Raspberry toujours, au niveau de l'icône 'VNC' dans la barre de tâches, ouvrir les options via le clic droit :
  - Dans l'onglet '**Sécurité**', choisir '**mot de passe VNC**'
  - Dans l'onglet '**utilisateurs et autorisations**', choisir '**utilisateur standard**' et cliquer sur '**mot de passe**' → fournir alors un mot de passe (qui peut être le même qu'en SSH pour faciliter les choses, à savoir pour nous '**Postexx**')
- ➔ Côté Windows, dans MobaXterm, créer une session VNC en fournissant l'adresse IP de la Raspberry et le mot de passe précédent :



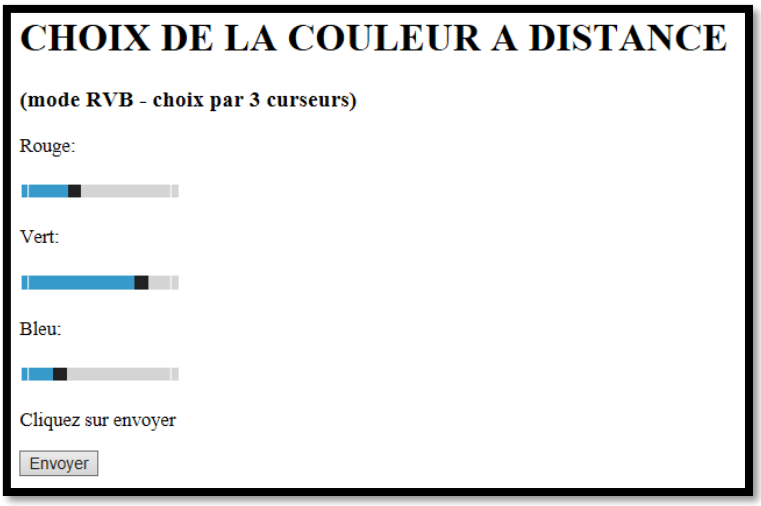
➔ Attendre quelques instants que le bureau se mette en place



Ces deux méthodes vous permettront donc de prendre le contrôle de votre Raspberry à distance sans avoir besoin de lui connecter des interfaces utilisateur comme l'écran, le clavier ou la souris.
Vous pouvez donc développer vos programmes Python sur votre poste Windows et les déployer sur la Raspberry via le réseau.

EXERCICE 4 : 'Ex04.py'

On va ici chercher à utiliser des 'sliders' sur la page web afin de pouvoir modifier à distance chacune des composantes Rouge/Vert/bleue de la matrice de la carte '**Sense Hat**'.
Voici l'aspect final de notre page web :



Le but dans ce module n'étant pas de vous initier au langage HTML, je vous fournis donc le code ci-dessous :

```
<html>
    <head>
        <title>COULEUR</title>
        <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
    </head>
    <body>
        <h1>CHOIX DE LA COULEUR A DISTANCE</h1>
        <h3>(mode RVB - choix par 3 curseurs)</h3>
        <form method="post" action="{{url_for('change')}}">
            <div class="slidecontainer">
                <p>Rouge:</p>
                <input type="range" min="0" max="255" class="slider" value={{valr}} name="rouge">
                <p>Vert:</p>
                <input type="range" min"0" max="255" class="slider" value={{valv}} name="vert">
                <p>Bleu:</p>
                <input type="range" min="0" max="255" class="slider" value={{valb}} name="bleu">
            <p>Cliquez sur envoyer</p>
            <input type="submit" name="switch" value="Envoyer"/>
            </div>
        </form>
    </body>
</html>
```

Vous l'enregistrerez sous le nom '**Ex04.hmtl**'.

On y retrouve la méthode '**post**' et des balises '**input**'. Mais au lieu d'être de type '**text**' comme dans l'exercice précédent, elles sont ici de type '**range**' ce qui correspond à une interface utilisateur avec un '**slider**'.
On a donc forcément 3 balises de 'slider' pour les trois couleurs. Le min et le max pour chacun correspond aux plages de variation des couleurs rouge, vert, bleu pour la matrice.
Leur nom ('**name**') va nous permettre de les identifier via les requêtes dans le programme python ('**request.form**').
Quant à leurs valeurs ('**value**'), elle est spécifiée avec des doubles accolades, ce qui signifie que ce sont des valeurs provenant du fichier Python (cf. Ex01 avec la température).

Pour la partie Python ('**Ex04.py**'), je vous laisse vous inspirer de tout ce qui a été fait précédemment pour l'écrire. Sachez simplement qu'il vous faut 3 requêtes dans votre fonction '**change()**', une par couleur et que le 'render_template' aura également 3 paramètres (Cf. Ex01 avec la température pour vous aider).

TUTO 4 : 'Tuto04.py'

Utilisation de Flask socket.io !! ??