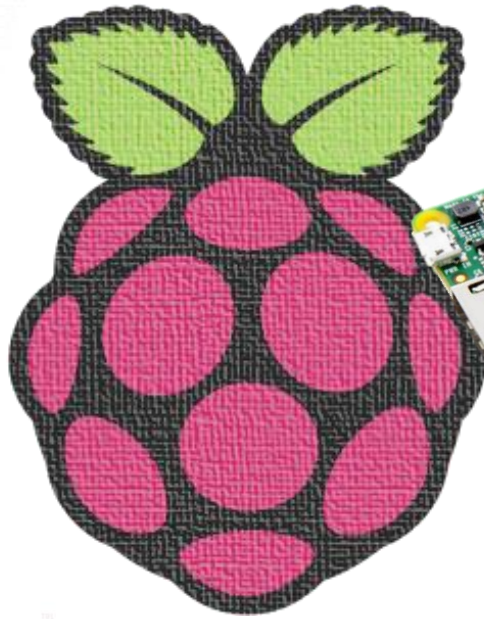


# Découverte de la Raspberry Pi



## Travaux pratiques Séance n°4

## Objectifs de la séance :

- Récupérer des informations depuis internet et les traiter
- Découvrir comment contrôler et programmer la Raspberry à travers un réseau local
- Installer un serveur sur la Raspberry et créer ses propres pages web

## 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
```

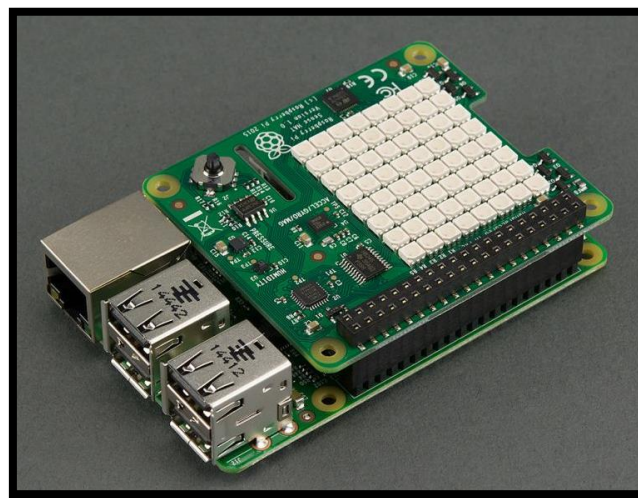
[rem : il n'y a pas d'espace dans ce lien, juste des '\_']

Une fois la tâche effectuée, vous devriez retrouver, en ouvrant un explorateur de fichiers, un dossier 'IUT\_IPE\_TP4' sous '/home/pi'.

## Recommandation importante concernant les programmes à écrire :

On utilisera comme d'habitude l'environnement de développement 'Geany' mais cette fois-ci, vous ne baserez pas vos programmes sur le template 'main\_IPE.py', vous partirez de zéro.

On va continuer pour ce TP d'associer la Raspberry avec la carte d'extension Sense Hat :



Installez-la (si ce n'est pas déjà fait !!) sur la Raspberry **avant** d'alimenter l'ensemble.

## EXERCICE 1 : 'Ex01.py'

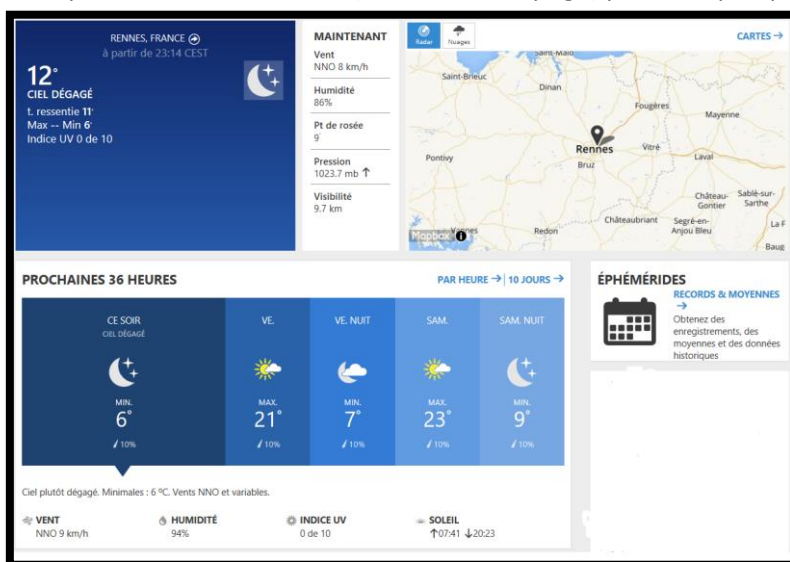
### Objectif de l'exercice :

Mise à part l'exercice n°11 du TP n°2, nous n'avons pas encore utilisé un des grands points forts de la Raspberry : sa connexion facile avec le reste du monde !!

Dans la première moitié de ce TP, on va donc exploiter cette capacité en créant un système permettant d'afficher les prévisions météo du jour sur la matrice de LEDs de la carte Sense Hat.

Il existe un nombre impressionnant de sites web fournissant des prévisions météo. Bon nombre d'entre eux fournissent des **API (Application Programming Interface** – « ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels » dicit Wikipedia) dans différents langages : C++, Java (JSON), Python, XML, ...

Nous allons utiliser celle du site historique '[www.weather.com/fr-FR](http://www.weather.com/fr-FR)'. Commencez par consulter la page web et regarder les prévisions pour la ville de RENNES (lien en bas de page) pour les quelques jours à venir :



Notez notamment pour aujourd'hui :

- La description
- L'icône associé
- La température maximale
- La température minimale

Les prévisions à 5 jours vous donnent un bon résumé de ce qui va nous être utile :

JOUR	DESCRIPTION	ÉLEVÉ / BAS	PRÉCIP.	VENT	HUMIDITÉ
CE SOIR 13 SEPT.	Ciel dégagé	--/6	10%	NNO 9 km/h	94%
VE. 14 SEPT.	Plutôt ensoleillé	21/7	10%	NO 12 km/h	66%
SAM. 15 SEPT.	Plutôt ensoleillé	23/9	10%	ENE 7 km/h	67%
DIM. 16 SEPT.	Peu nuageux	26/10	10%	S 9 km/h	55%
LU. 17 SEPT.	Ensoleillé	28/16	10%	SE 10 km/h	61%
MAR. 18 SEPT.	Ensoleillé	28/14	0%	SSO 25 km/h	64%

La première étape va être d'installer la librairie Python qui va nous permettre d'exploiter l'API fournie par ce site web. Il s'agit de **'pywapi'**.

Habituellement, dans les TP précédents, lorsque l'on avait besoin d'une librairie, on utilisait les serveurs officiels via les commandes `'sudo apt-get'` ou `'sudo pip'`. Mais pour **'pywapi'**, elle n'est pas présente sur les serveurs officiels (comme énormément d'autres librairies, notamment celles développées par les makers ou les hobbyistes !!). Cela va donc nous donner l'occasion de voir comment installer une librairie fournie par `'github'` ou autre.

Les étapes sont les suivantes :

- Télécharger l'archive via la page web suivante : <https://code.google.com/archive/p/python-weather-api/> (fichier avec extension `'.tar.gz'`)
- Si vous avez utilisé le navigateur `'chromium'`, elle va se trouver dans le dossier **'Downloads'**
- Faire un clic droit sur le fichier archive et choisir **'Extraire ici'**
- Allez alors dans ce dossier et ouvrez un terminal à cet emplacement (via le menu **'Outils'**)
- Entrez alors la commande suivante : **`$ sudo python setup.py install`**
- Et voilà, au bout de quelques instants, votre librairie est installée

Pour utiliser cette librairie Python afin de récupérer les prévisions météo, il nous faudra connaître le code identifiant la ville de RENNES (**'location\_id'**).

Créer donc un premier programme **'Ex01.py'** qui va nous fournir ce code.

#### Informations utiles :

- Pour utiliser la librairie, on a juste besoin de l'importer au tout début du fichier : **`'import pywapi'`**
- Si l'on regarde la partie `'weather.com'` de la documentation de la page web `'pywapi'`, on y lit que la méthode **`'get_location_ids()'`** permet de trouver le code identifiant que l'on recherche. Vous l'appellerez donc par : **`'pywapi.get_location_ids("ville")'`**

**Une fois votre programme exécuté, notez bien le code fourni pour la ville de RENNES.**

Pour contrôler, jeter donc un coup d'œil dans la barre d'adresse correspondant à la page web des prévisions météo de Rennes.

## EXERCICE 2 : 'Ex02.py'

#### Objectif de l'exercice :

Maintenant que l'on connaît le code identifiant la ville de Rennes, on va pouvoir récupérer les prévisions météo pour cette localisation.

Le but de ce second programme va être de récupérer ces infos et de les afficher dans leur format brut d'échange dans un terminal.

#### Informations utiles :

- Il va falloir utiliser la méthode **`'get_weather_from_weather_com("code identifiant")'`** pour récupérer ces prévisions dans un objet. Cet objet, vous allez l'appeler **`'previsions_meteo'`**.
- Utilisez un simple **`'print'`** pour afficher cet objet.

Exécutez votre programme et regardez bien la structure et la quantité des données reçues. On peut également remarquer que tous les mots sont en anglais.



**Remarque :**

En fait, le site web renvoie ces prévisions au format **JSON** ('**Javascript Objet Notation**'). A notre niveau de connaissances, le format JSON des données est très proche d'une structure python que l'on a déjà rencontrée : **le dictionnaire**. En effet, on voit apparaître des accolades, des clefs et des valeurs (séparées par ':').

En exploitant cette similitude, on peut utiliser une librairie native de Python ('**pprint**') qui rend l'affichage des dictionnaires beaucoup plus lisible. Ouvrez, exécutez le fichier '**Ex02\_variante.py**' et observez le résultat obtenu. Retrouvez-vous les prévisions à 5 jours de l'exercice n°1 ?

**Faire contrôler votre programme par un enseignant.**

## **EXERCICE 3 : 'Ex03.py'**

**Objectif de l'exercice :**

On va poursuivre notre récupération de prévisions météo en extrayant dans le flot de données reçues : la date d'aujourd'hui, la tendance, la température maximale, la température minimale et l'icône météo.

Ces informations seront clairement affichées dans un terminal.

**Informations utiles :**

- La structure des données reçues est assez complexe. Il existe peut-être une librairie de plus haut niveau pour les extraire, mais je n'en ai pas connaissance pour le moment. Il va donc falloir accéder à des informations qui sont imbriquées les unes dans les autres.  
Le premier caractère des données reçues est '{'. Il s'agit donc d'un dictionnaire qui comporte des clefs et des valeurs.  
Or pour un dictionnaire simple de type :  
**Dic = {'clef1' : '12', 'clef2' : '56', 'clef3' : '87'}**  
Si l'on fait :  
**Print Dic['clef2']**  
On obtient :  
**56**
- Les prévisions jour après jour s'obtiennent avec la clef '**forecasts**'.  
Essayez donc la commande suivante :  
**pprint(previsions\_meteo['forecasts'])**
- Le premier caractère qui apparaît est '['. Il s'agit donc d'une liste. La liste n'a pas de clef, juste un indice comme un tableau.  
Chaque élément de cette liste est à nouveau un dictionnaire dont la première clef est le mot 'date'.  
On peut donc en déduire que chaque élément de la liste correspond aux prévisions pour un jour donné.  
L'indice 0 va correspondre à aujourd'hui.  
L'indice 1 à demain.  
L'indice 2 à après-demain.  
Et ainsi de suite jusqu'à l'indice 4 car ce sont des prévisions à 5 jours !!  
Essayez donc la commande suivante maintenant :  
**pprint(previsions\_meteo['forecasts'][0])**  
Est-ce qu'il s'agit bien des informations concernant la météo d'aujourd'hui ?
- Continuons notre démarche. Le premier caractère de ce que l'on vient d'afficher est à nouveau '{'. Il s'agit donc encore d'un dictionnaire qui comporte des clefs.  
Les clefs sont ici : '**date**', '**day**', '**day\_of\_week**', '**high**', '**low**', '**night**', ...

Et donc pour afficher à la température basse sur la journée, il suffit d'entrer la commande :

```
pprint(previsions_meteo['forecasts'][0]['low'])
```

rem : il s'agit bien d'une chaîne de caractères

- En utilisant cette démarche, compléter votre programme pour afficher :
  - La date (jour de la semaine + date)
  - La tendance de la journée (clef '**text**')
  - La température la plus haute
  - La température la plus basse
  - Le numéro de 'icône météo (clef '**icon**')

**Faire contrôler votre programme par un enseignant.**

## EXERCICE 4 : 'Ex04.py'

### Objectif de l'exercice :

On va améliorer le programme précédent sur un point précis : rendre son affichage lisible par toute personne non familière à la langue de Shakespeare !!

En effet, le jour de la semaine et la tendance sont exprimées en anglais. On va corriger ceci.

### Informations utiles :

- On va pour cela utiliser les services de '**Google translate**'. Il faut pour cela installer la librairie Python correspondante via le terminal : **sudo pip install googletrans**
- L'utilisation de cette librairie est très simple malgré le fait qu'elle puisse traduire à peu près tout ce que l'on veut et dans n'importe quelle langue !! Cependant, gardez en tête qu'il s'agit d'un service cloud et qu'une connexion internet est donc nécessaire. Consultez la page web suivante (rubrique '**basic usage**') pour découvrir comment l'utiliser : <https://py-googletrans.readthedocs.io/en/latest/>
- La réponse fournie par la méthode 'translate' est composée de plusieurs éléments comme la langue source et celle de destination, la traduction, la prononciation et l'indice de confiance. Pour nous, c'est l'attribut 'text' qui nous intéresse car il contient la traduction. La rubrique '**advanced usage**' de la page web vous donnera un indice sur la façon dont on le récupère.
- Modifiez donc le programme précédent pour afficher les mêmes informations dans le terminal mais en ayant traduit le jour de la semaine et la tendance météo du jour.

## EXERCICE 5 : 'Ex05.py'

### Objectif de l'exercice :

On va maintenant exploiter ce que l'on a utilisé dans le TP n°3 pour afficher toutes ces informations sur la matrice de la carte Sense Hat.

### Informations utiles :

Compléter le programme précédent pour afficher les prévisions du jour sur la matrice plutôt que dans le terminal :

- Commencez par afficher la date du jour des prévisions (méthode '**show\_message()**')
- Affichez ensuite l'icône. Pour cela, le site [www.weather.com](http://www.weather.com) transmet un numéro que vous avez affiché dans les exercices précédents. Il va maintenant falloir faire correspondre ce numéro à une

image et l'afficher. Les images seront forcément très simplifiées due à la résolution de la matrice (8x8). Dans le dossier '**images**' du TP4, vous trouverez certaines d'entre elles (elles portent le même numéro que celui des prévisions). Affichez donc l'image si elle existe sinon vous afficherez l'image '**error.png**'.

- Affichez ensuite la tendance, puis la température basse et enfin la température haute.
- Recommencez l'ensemble de l'affichage dans une boucle infinie.

**Faire contrôler votre programme par un enseignant.**

## EXERCICE 6 : 'Ex06.py'

### Objectif de l'exercice :

On va ajouter une dernière modification à ce terminal de prévisions météo. On va ajouter de la synthèse vocale comme celle de l'exercice n°8 du TP1 (service Google gTTS en ligne).

### Informations utiles :

Pour finir donc, compléter le programme précédent pour annoncer avec l'aide du service TTS de Google :

- La tendance
- La température basse
- La température haute

### Remarque :

Il ne serait pas compliqué d'améliorer encore un peu notre programme pour utiliser le joystick présent sur la carte afin de pouvoir naviguer entre les différentes informations à afficher et surtout pour pouvoir modifier le jour des prévisions affichées (par défaut, on reçoit les prévisions sur 5 jours et nous n'avons géré que la prévision du jour en cours jusqu'à présent). Mais, il y a encore plein d'autres choses intéressantes sur lesquelles on va se pencher maintenant...

## INTERMEDE :

On va juste faire une petite parenthèse informative entre deux exercices.

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.

## Utilisation de MobaXterm sous Windows :

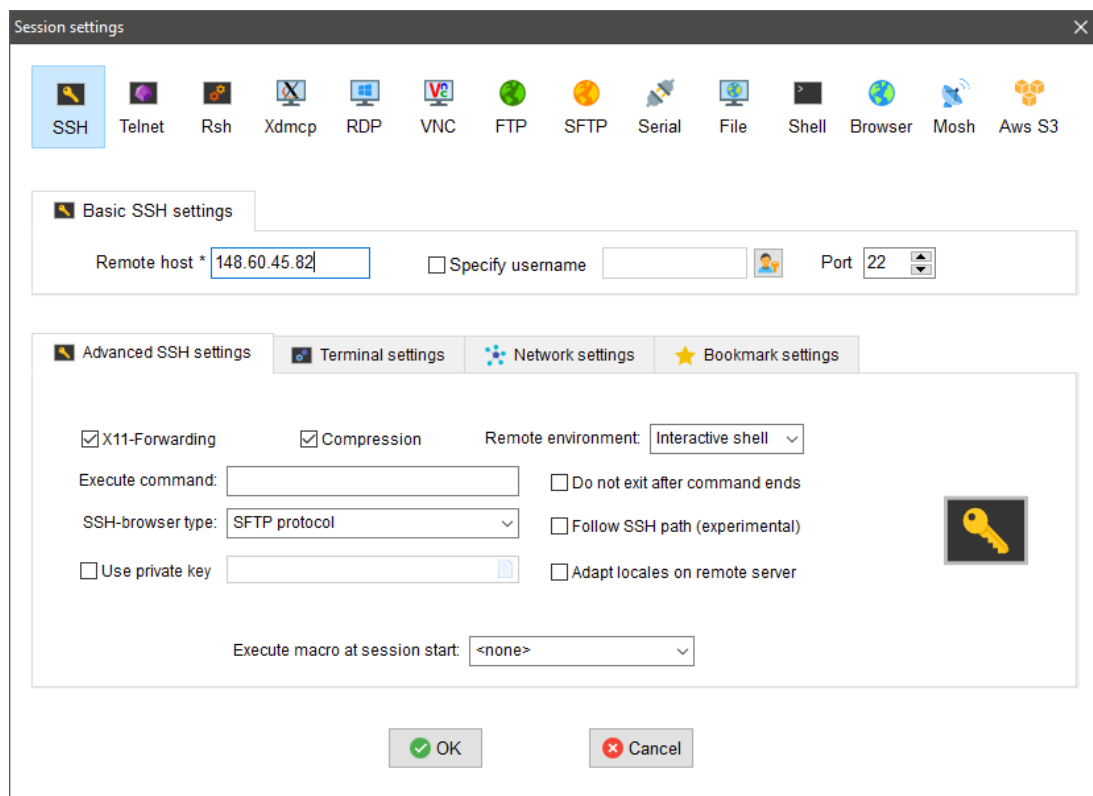
Le premier mode que l'on va découvrir est le '**SSH**', qui permet de récupérer la console/terminal de la Raspberry sous Windows, et de pouvoir également ouvrir la fenêtre d'une application à distance.

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 :

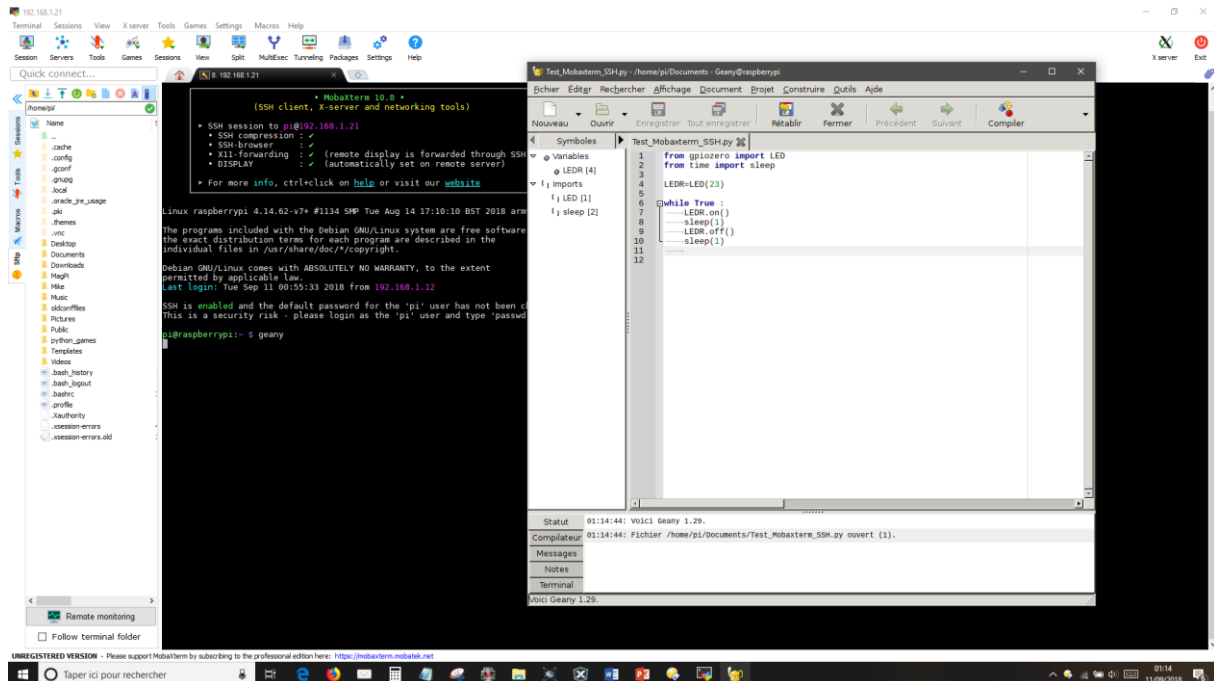
- ➔ Valider le 'SSH' dans les options de configuration de la Raspberry (Cf. TP n°1)
- ➔ Côté PC, sous MobaXterm, créer une session en fournissant l'adresse IP de la Raspberry à atteindre :



- ➔ 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.
- ➔ Faites 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 : **RPI\_xx** (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 '**geany**'. La fenêtre de Geany s'ouvrira alors dans Windows (enregistrement, exécution de programme, CTRL+C, ... tout fonctionne) (ATTENTION toutefois à ne pas faire '**sudo geany**' car on ne peut lancer des commandes qu'avec les droits de l'utilisateur SSH soit '**pi**' ici)

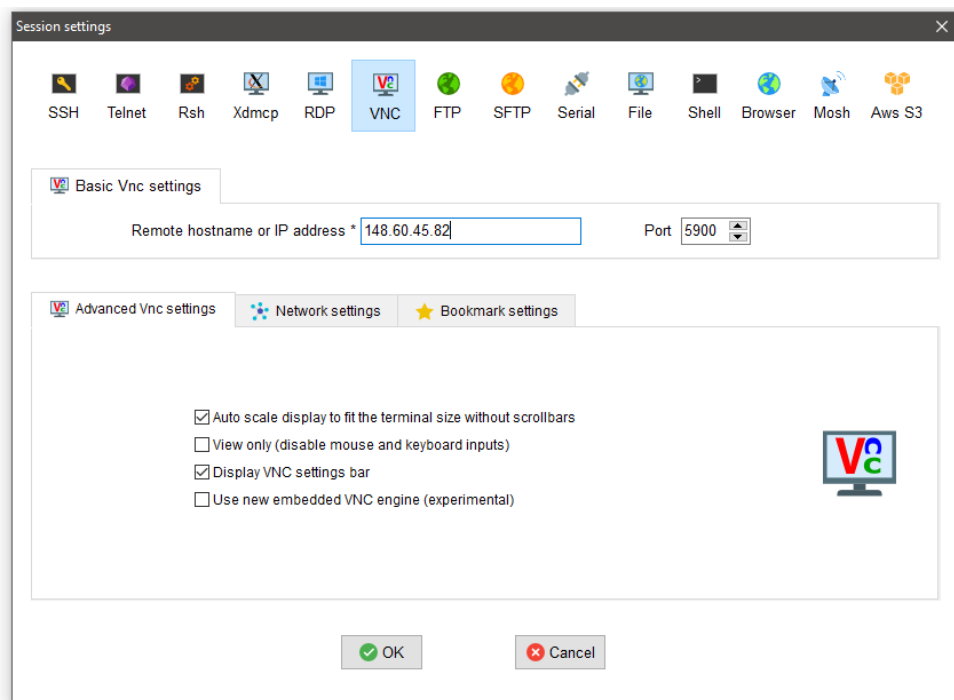


L'autre mode que l'on va décrire ici est le mode '**VNC**' (Virtual Network Computing).

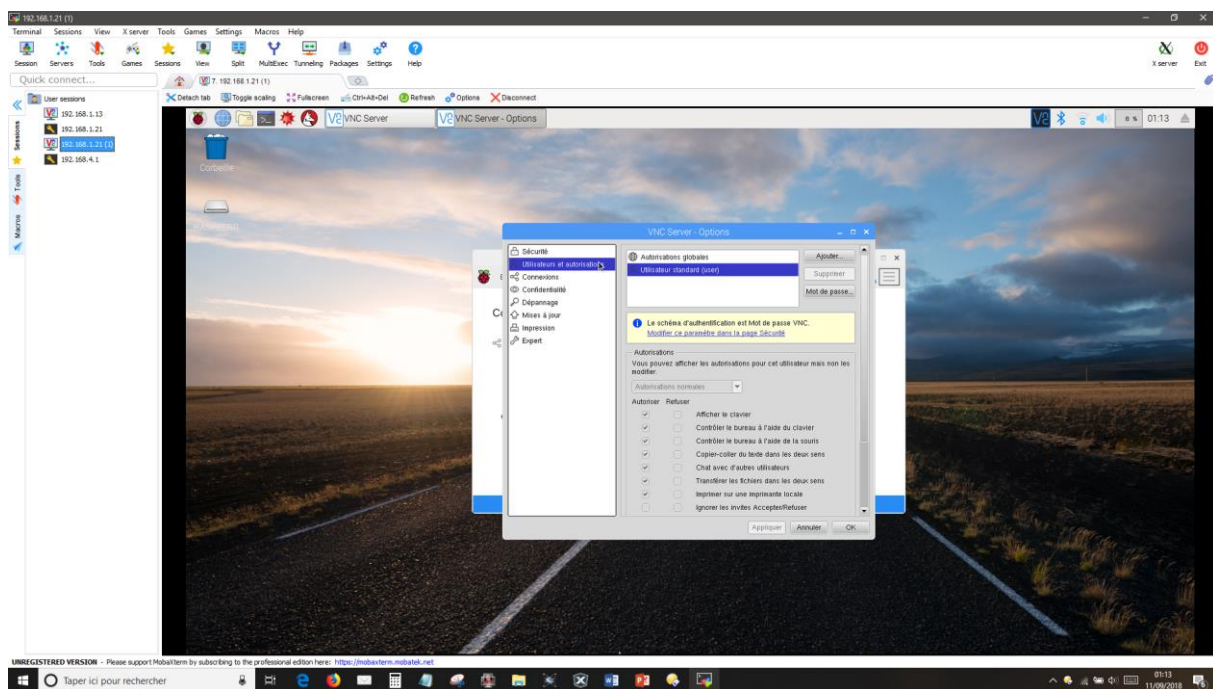
« C'est un système de visualisation et de contrôle de l'environnement de bureau d'un ordinateur distant. Il permet au logiciel client VNC de transmettre les informations de saisie du clavier et de la souris à l'ordinateur distant, possédant un logiciel serveur VNC à travers un réseau informatique » (Source Wikipedia).

#### En mode VNC :

- ➔ 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 '**RPI\_xx**')
- ➔ 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

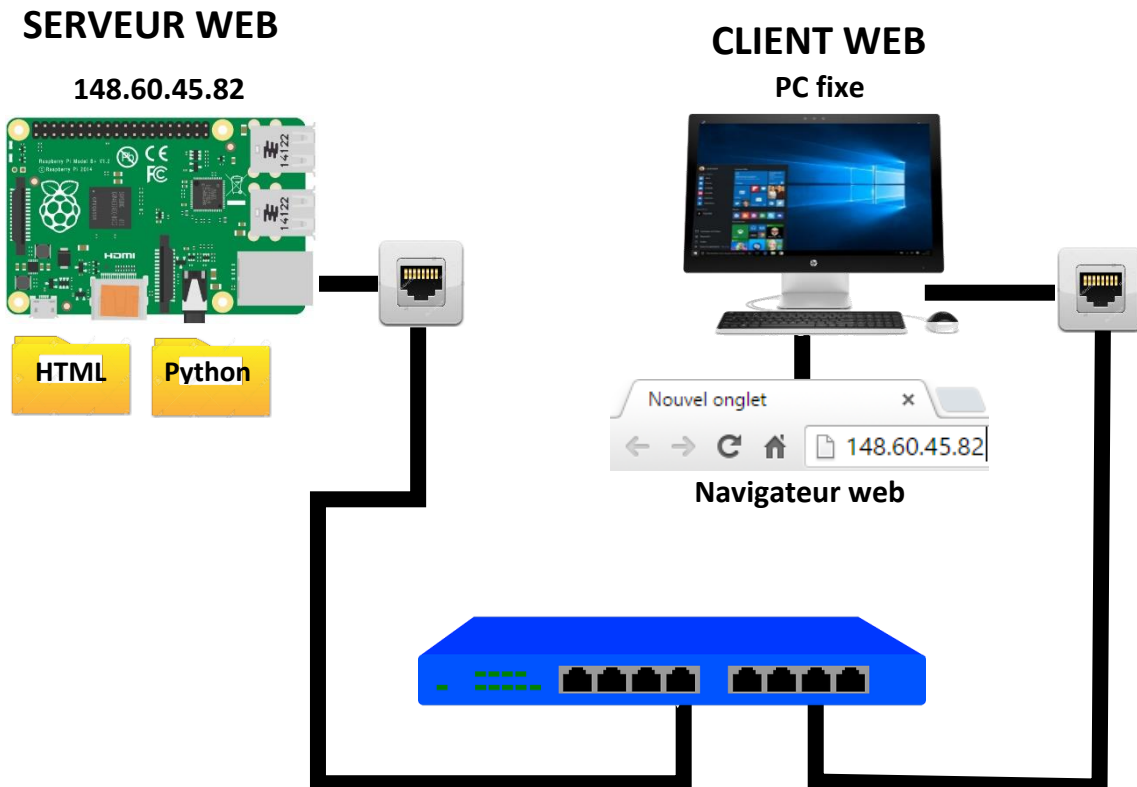


Ces deux méthodes vous permettront donc de prendre le contrôle de votre Raspberry à distance sans avoir besoin de lui connecter des interfaces utilisateur comme l'écran, le clavier ou la souris. Vous pouvez donc développer vos programmes Python sur votre poste Windows et les déployer sur la Raspberry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

## EXERCICE 07 : 'Ex07a.py' & 'Ex07b.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, mais permettra également à un script python d'être exécuté.

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.

La première étape consiste donc à télécharger et installer un serveur web. On va utiliser le package '**Flask**' (<http://flask.pocoo.org/>) :



‘Flask’ est installé en natif avec l’OS Raspbian Jessie. Toutefois, mieux vaut le vérifier et le cas échéant le mettre à jour. Deux commandes sont nécessaires dans un terminal pour cette vérification :

**\$ sudo pip install flask**

Ces deux opérations devraient vous prendre au maximum 2 à 3 minutes, voire moins si la version est à jour. A partir de là, on va pouvoir écrire notre programme Python et notre page HTML. Commencez par écrire dans ‘Geany’ le code suivant avec le nom de fichier ‘Ex07a.py’ :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# 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
if __name__ == '__main__' :
    serveur_web.debug = False
    serveur_web.run(host="0.0.0.0")
```

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

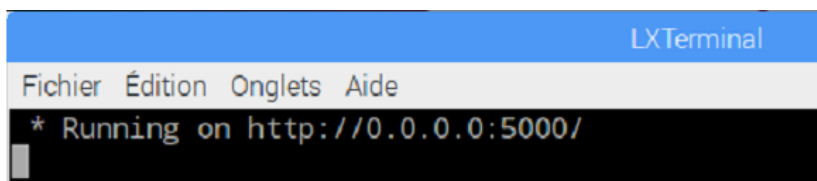
- ‘@serveur\_web.route(‘chemin\_URL’)’ : le code qui suit sera exécuté lorsque dans le navigateur client, on se connectera à l’adresse ‘adresse\_IP\_serveur/chemin\_URL’. Cela peut être du code python. Pour le moment, on ne renvoie juste que l’affichage d’un texte fixe.
- ‘if \_\_name\_\_ == ‘\_\_main\_\_’’: ce test est vrai uniquement si le programme python est exécuté directement, c’est-à-dire s’il n’est pas appelé par un autre programme, ce qui sera le cas ici.
- ‘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 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 'Geany', vous devriez voir apparaître dans le terminal le message suivant :



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 pour interroger le serveur tournant sur la Raspberry.

Ouvrez donc le navigateur côté Raspberry et testez les adresses suivantes :



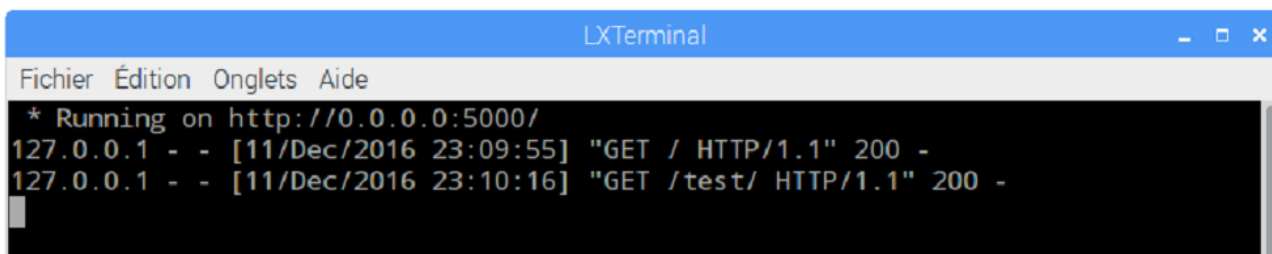
➔ Affiche la page renvoyée par la fonction 'racine()' en mode local



➔ 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.**

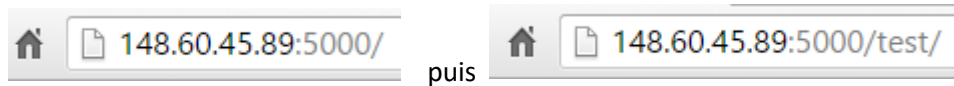
Dans le terminal, vous devriez notamment voir les différentes requêtes effectuées auprès de ce serveur durant son fonctionnement :



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



Exemple avec la Raspberry du poste n°9 :



**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 '**Ex07a.py**' ait été lancé sur la Raspberry interrogée.

Pour le moment, on a donc vu comment afficher du texte fixe 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.

Pour simplifier l'écriture du code python, on va utiliser ce que l'on appelle des '**templates**' HTML que '**Flask**' va générer en y intégrant des variables dynamiques de notre code python.

Ecrivez le code suivant dans '**Geany**' avec le nom de fichier '**Ex07b.py**' :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# 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('index.html', poste =socket.gethostname(), heure =chaine_heure)

# PROGRAMME PRINCIPAL
if __name__ == '__main__' :
    serveur_web.debug = False
    serveur_web.run(host="0.0.0.0")
```

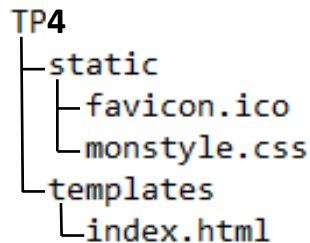
Là aussi, quelques remarques s'imposent pour aider à la compréhension :

- '**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 'index.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**' (rappelez-vous qu'il s'agit d'un éditeur de texte puissant et pas d'un environnement de développement dédié uniquement à python. On

peut donc y programmer dans n'importe quel langage !!), créez un nouveau fichier que vous nommerez 'index.html'.

Pour le sauvegarder, vous devrez respecter une certaine arborescence, faute de quoi, votre page ne s'affichera pas correctement :



Tapez le code suivant dans le fichier 'index.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>
```

Quelques explications :

- ➔ 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
  - <br/> : 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}} comme ici pour 'poste' et 'heure'. Ceci se fait grâce à une compatibilité de 'Flask' avec 'Jinja'.
- ➔ Le texte entre les balises <link> permet l'affichage de l'icône de la page dans l'onglet du navigateur.

Vous pouvez tester comme tout à l'heure l'exécution du code ('Ex07b.py') en mode local puis en mode distant dans le navigateur du PC.

**Faire contrôler votre programme par un enseignant.**

## EXERCICE 8 : 'Ex08.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 '**Ex08.py**' et la page HTML '**temperature.html**' permettant de réaliser ce qui vous est demandé.

**Faites 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 qui sort du cadre de ces TPs.

## EXERCICE 9 : 'Ex09.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 '**commande\_on\_off.html**' (à stocker dans le dossier '**templates**') et insérez-y le code suivant :

```
<html>
  <head>
    <title>MATRICE</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 un nouveau fichier python que vous nommez **‘Ex09.py’** et copiez-y le code suivant :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# 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('commande_on_off.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('commande_on_off.html')

# PROGRAMME PRINCIPAL
if __name__ == '__main__' :
    serveur_web.debug = False
    serveur_web.run(host="0.0.0.0")
```

Testez alors le bon fonctionnement de votre programme et **faites-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>MATRICE</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 v2</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 10 : 'Ex10.py'

On va enfin, chercher à reproduire ce que l'on a fait dans le programme n°2 du 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 dans un fichier que nous nommerons '**affichage\_message.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 '**Geany**', créez un nouveau fichier '**Ex11.py**' et recopiez-y le code ci-dessous en le complétant :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# 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()

# Définition des couleurs
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('affiche_message.html')

@serveur_web.route('/change/', methods=['POST'])
def change():
    if request.method == 'POST' :
        message_a_afficher = # A COMPLETER
        # Affichage du message
        sense.show_message(# A COMPLETER)
        # Efface la matrice
        # A COMPLETER
    return render_template('affiche_message.html')

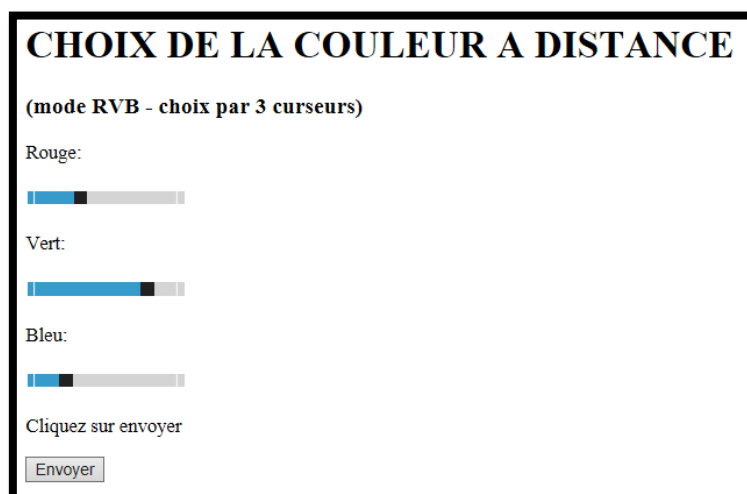
# PROGRAMME PRINCIPAL
if __name__ == '__main__' :
    serveur_web.debug = False
    serveur_web.run(host="0.0.0.0")
```

Pour ce code Python, vous devez compléter la récupération du ‘message’ et son affichage.

**Faire contrôler votre programme par un enseignant.**

## EXERCICE 11 : ‘Ex11.py’ (optionnel)

Maintenant que l’on est parvenu à piloter la matrice à distance à travers plusieurs exercices, on va chercher maintenant à modifier la couleur de la matrice à travers le réseau. On va pour cela utiliser des ‘sliders’ :



**CHOIX DE LA COULEUR A DISTANCE**

(mode RVB - choix par 3 curseurs)

Rouge:

Vert:

Bleu:

Cliquez sur envoyer

Envoyer

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 '**couleur\_matrice.html**'.

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. Ex08 avec la température).

Pour la partie Python ('**Ex11.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. Ex08 avec la température pour vous aider).