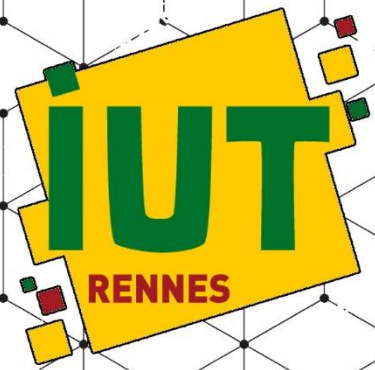
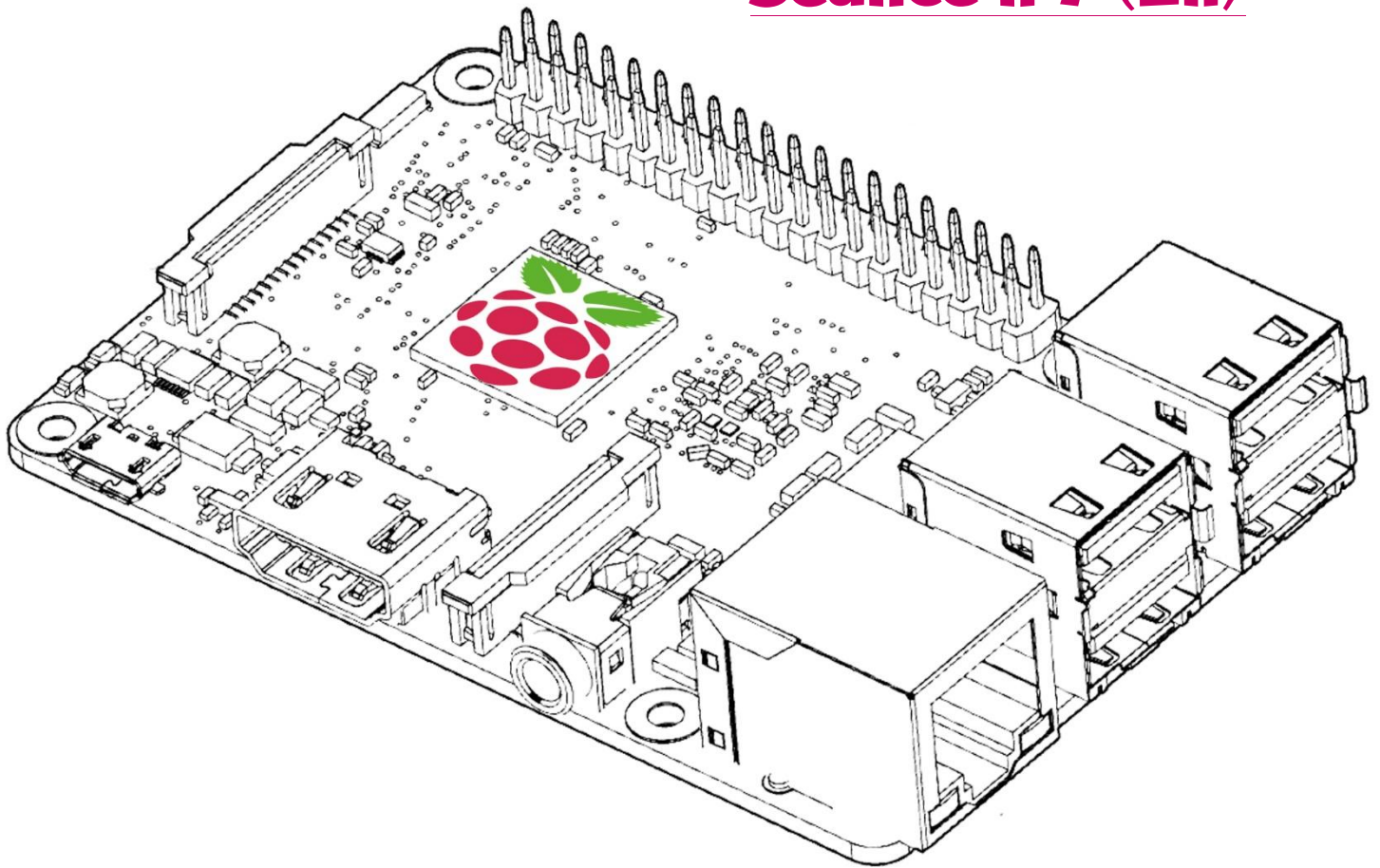


AT31 - Module IPE



Séance n°7 (2h)



Utilisation d'APIs
partie 2

Michaël BOTTIN

Objectifs de la séance :

- Poursuivre la récupération de données sur le web via un fichier JSON
- Récupération de données sur le web via des APIs de très haut niveau

Récupération des fichiers utiles pour cette séance :

Comme pour le TP précédent, il vous faut récupérer les fichiers utiles pour l'écriture de vos programmes. Vous allez les récupérer depuis **Github**.

Pour cela, tapez la commande suivante dans le terminal :

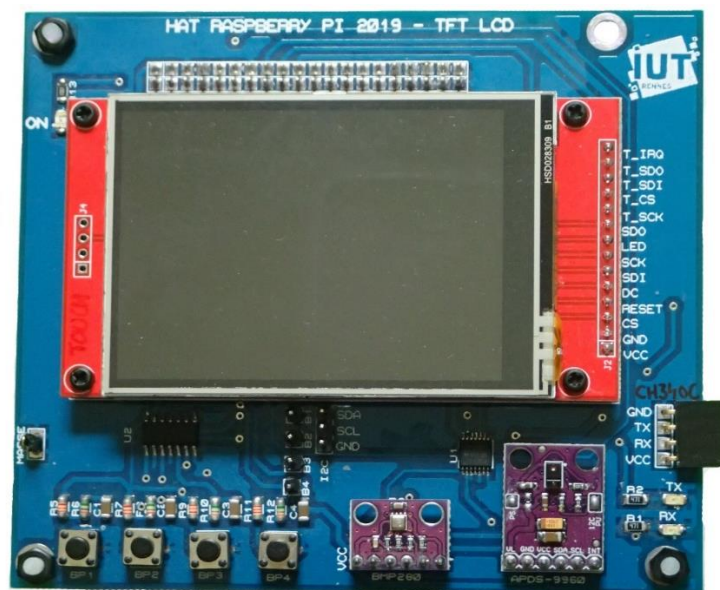
```
$ git clone https://github.com/MbottinIUT/IUT\_IPE\_TP7.git
```

[remarque : 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_TP7' sous '/home/pi'.

Carte que l'on va utiliser :

On va continuer d'utiliser la carte d'extension qui dispose d'un écran LCD :



Pensez à vérifier que les interfaces 'I2C' et 'SPI' sont bien activées sur la Raspberry (via la fenêtre des préférences de la Raspberry via le menu 'démarrer' puis l'onglet 'interfaces').

Préambule :

On va prendre juste un peu de temps pour parler de la structuration des données en Python. Il existe trois principales formes pour la structuration des données en Python :

- Les listes
- Les tuples
- Les dictionnaires

Vous les avez déjà toutes utilisées sans le savoir. Toutefois, pour mieux comprendre les exercices de cette séance et de la suivante, il convient de revenir sur quelques points.

Les listes :

C'est la structuration de données qui ressemble le plus à ce que vous connaissez en C à savoir les tableaux.

Une liste de taille N voit ses éléments indexés de 0 à N-1.

Vous avez, par exemple, utilisé des listes pour définir vos couleurs lorsque l'on fait défiler un texte sur la matrice de la carte Sense Hat :

show_message

Scrolls a text message from right to left across the LED matrix and at the specified speed, in the specified colour and background colour.

Parameter	Type	Valid values	Explanation
<code>text_string</code>	String	Any text string.	The message to scroll.
<code>scroll_speed</code>	Float	Any floating point number.	The speed at which the text should scroll. This value represents the time paused for between shifting the text to the left by one column of pixels. Defaults to <code>0.1</code> .
<code>text_colour</code>	List	<code>[R, G, B]</code>	A list containing the R-G-B (red, green, blue) colour of the text. Each R-G-B element must be an integer between 0 and 255. Defaults to <code>[255, 255, 255]</code> white.
<code>back_colour</code>	List	<code>[R, G, B]</code>	A list containing the R-G-B (red, green, blue) colour of the background. Each R-G-B element must be an integer between 0 and 255. Defaults to <code>[0, 0, 0]</code> black / off.

Returned type	Explanation
None	

```

from sense_hat import SenseHat

sense = SenseHat()
sense.show_message("One small step for Pi!", text_colour=[255, 0, 0])

```

Contrairement au C, lorsque l'on crée une liste (soit un tableau d'éléments indexés) en python, on n'a pas besoin de spécifier sa taille :

Liste1 = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Et comme on n'a pas besoin non plus de préciser le type des données à y stocker, on peut mixer ce que l'on veut dans une liste :

Liste2 = ["nom", 15, True, 5.6]

On accède aux éléments simplement en précisant leur index : **Liste1[5]** ou **Liste2[0]**

On peut également connaître la taille d'une liste par l'instruction : **len(Liste1)**

De plus, une liste étant un objet en python, on accède à différents attributs et méthodes :

- **append(x)** : ajoute l'élément x à la fin de la liste
- **insert(i,x)** : insert l'élément x à l'index i dans la liste
- **remove(x)** : supprime l'élément x de la liste
- **reverse()** : inverse l'ordre des éléments de la liste
- **count(x)** : compte le nombre d'apparition de l'élément x dans la liste
- **index(x)** : renvoie la position de l'élément x dans la liste
- **clear()** : efface le contenu de la liste
- (...)

Le contenu d'une liste n'est donc pas figé et peut être modifié tout au long de son utilisation.

Les tuples :

C'est une structuration de données qui ressemble en C également à un tableau à condition que la déclaration de ce dernier soit précédée du mot-clef '**const**' (pour le compilateur XC8).

En effet, les tuples sont comme les listes mais leurs éléments sont immuables, c'est-à-dire que l'on ne peut pas les modifier.

Sur la carte Sense Hat, le joystick utilise les tuples pour renvoyer les actions effectuées par l'utilisateur :

Joystick

InputEvent

A tuple describing a joystick event. Contains three named parameters:

- **timestamp** - The time at which the event occurred, as a fractional number of seconds (the same format as the built-in **time** function)
- **direction** - The direction the joystick was moved, as a string (**"up"**, **"down"**, **"left"**, **"right"**, **"middle"**)
- **action** - The action that occurred, as a string (**"pressed"**, **"released"**, **"held"**)

This tuple type is used by several joystick methods either as the return type or the type of a parameter.

Un tuple de longueur N a également ses éléments qui sont indexés de 0 à N-1.

On accède aux éléments de la même façon que pour une liste. Cependant, toutes les méthodes des listes qui ajoutent ou suppriment des éléments ne sont pas utilisables avec des tuples puisque le contenu est immuable.

On les distingue des listes car ils utilisent des **()** au lieu des **[]** :

Tuple1 = (255, 145, 0)

Tuple2 = ("dupont", "armand", 53)

Un tuple peut même contenir des listes.

Les tuples permettent les affectations multiples du type :

a, b = (12, 56)

a vaut alors 12 et b 56.

Lorsqu'à la fin d'une fonction, on désire renvoyer plusieurs variables/objets (ce qui n'est pas possible en C, d'où l'utilisation des pointeurs), elles sont renvoyées sous la forme d'un tuple. Voici un court exemple pour comprendre le principe : <https://www.geeksforgeeks.org/g-fact-41-multiple-return-values-in-python/>

Les dictionnaires :

C'est une structuration de données qui est également sous la forme d'un tableau mais les éléments ne sont pas indexés, ils sont identifiés par une clef. Leur ordre au sein du dictionnaire n'a donc aucune importance.

Les dictionnaires sont identifiables par leur déclaration avec des accolades : {}

Voici un exemple de dictionnaire en python :

Data = {"nom" : "Dupont", "age" : 58, "commune" : "Rennes"}

Chaque élément est sous ce format : **CLEF : VALEUR**

Pour afficher la commune, il suffit d'écrire :

Print (Data["commune"])

Ce qui nous donnera :

Rennes

Des attributs et des méthodes spécifiques sont disponibles en python pour les objets de type dictionnaires.

Quand vous avez utilisé l'accéléromètre de la carte Sense Hat, la méthode 'get_accelerometer_raw()' renvoyait un dictionnaire :

get_accelerometer_raw

Gets the raw x, y and z axis accelerometer data.

Returned type	Explanation
Dictionary	A dictionary object indexed by the strings <code>x</code> , <code>y</code> and <code>z</code> . The values are Floats representing the acceleration intensity of the axis in Gs.

```
from sense_hat import SenseHat

sense = SenseHat()
raw = sense.get_accelerometer_raw()
print("x: {x}, y: {y}, z: {z}".format(**raw))

# alternatives
print(sense.accel_raw)
print(sense.accelerometer_raw)
```

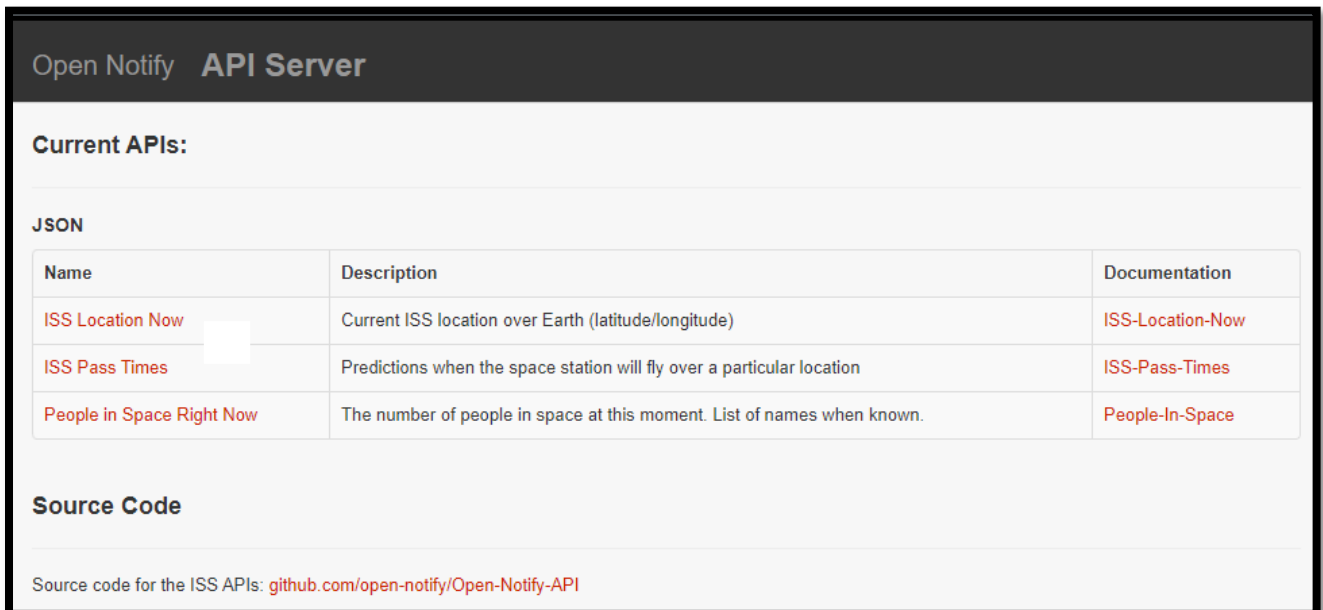
Le format de l'objet 'raw' de l'exemple ci-dessus est :

{'x' : 0.235689, 'y' : 0.99996541, 'z' : 0.5463298}

RECUPÉRATION DE DONNÉES ISSUE DU WEB VIA UN FICHIER JSON

TUTO 1 : 'Tuto01.py'

On va commencer avec un exemple très simple. Lancez un navigateur et ouvrez la page suivante :
<http://api.open-notify.org>



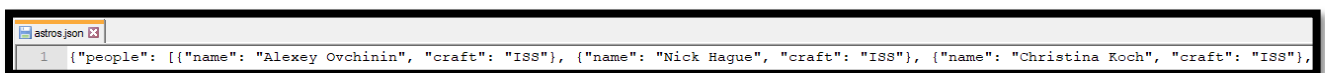
Name	Description	Documentation
ISS Location Now	Current ISS location over Earth (latitude/longitude)	ISS-Location-Now
ISS Pass Times	Predictions when the space station will fly over a particular location	ISS-Pass-Times
People in Space Right Now	The number of people in space at this moment. List of names when known.	People-In-Space

Source code for the ISS APIs: github.com/open-notify/Open-Notify-API

Cette page permet de récupérer quelques informations concernant l'ISS, la Station Spatiale Internationale.

On va s'intéresser au 3^{ème} lien du tableau qui va nous fournir un fichier au format JSON qui contiendra le nombre de personnes dans la station et leurs noms.

Cliquer sur le lien '**People in Space Right Now**' pour afficher le contenu du fichier '**astros.json**' dans le navigateur.



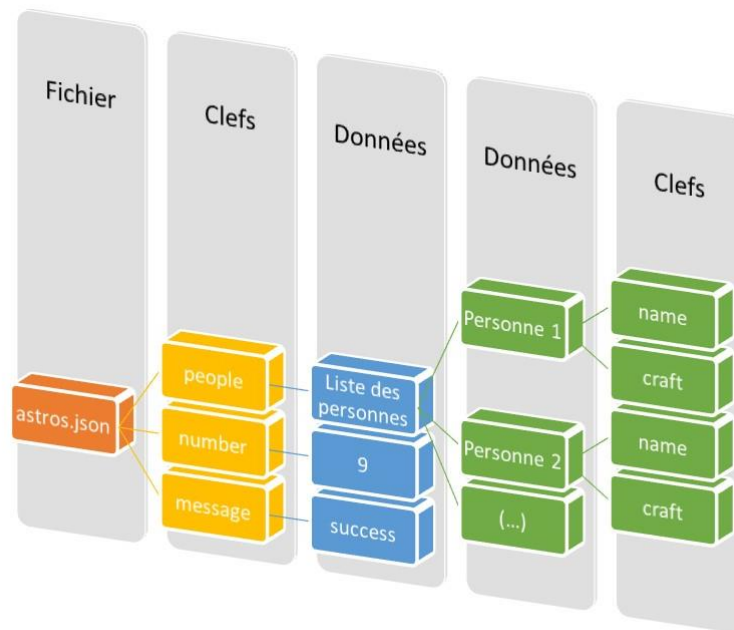
```
{"people": [{"name": "Alexey Ovchinin", "craft": "ISS"}, {"name": "Nick Hague", "craft": "ISS"}, {"name": "Christina Koch", "craft": "ISS"}]}
```

Ce format **JSON** ('**Javascript Objet Notation**'), tout comme CSV, RSS, XML est un format d'échanges de données sur le web. A notre niveau de connaissances, le format JSON est très proche d'une structure de donnée en python que l'on vient d'évoquer : **le dictionnaire**. En effet, on voit apparaître des accolades, des clefs et des valeurs (séparées par ':').

Mais à l'intérieur de ce dictionnaire, on voit également des crochets qui caractérisent les listes.

Plusieurs types peuvent donc s'imbriquer pour créer des objets complexes mais à haute valeur informative !!

Si l'on tente de représenter ces données sous forme d'une arborescence, voici à quoi cela pourrait ressembler :



Sachant que 'Personne 1', 'Personne 2',... sont elles aussi des dictionnaires contenant le nom de la personne et le nom de l'objet spatial où elles sont.

Supposons que l'objet '**personnes_ISS_JSON**' contienne tout le fichier JSON que l'on va récupérer. Ainsi on accède au nombre de personnes dans l'ISS par :

personnes_ISS_JSON['number']

On accédera au dictionnaire de la personne 1 par :

personnes_ISS_JSON['people'][0]

Et on accédera au nom de la 3^{ème} personne par :

personnes_ISS_JSON['people'][2]['name']

Voyons maintenant comment tout cela peut se coder en Python.

Ouvrez le fichier '**Tuto01.py**' dans Thonny :

```

1  # Importation des librairies
2  import requests
3  import pprint
4
5  # www.open-notify.org
6  # Récupération d'un fichier JSON correspondant au nombre de personne dans l'ISS
7  url = 'http://api.open-notify.org/astros.json'
8  personnes_ISS = requests.get(url)
9
10 # Récupération de flux JSON au sein de la requête effectuée et affichage
11 personnes_ISS_JSON = personnes_ISS.json()
12 print ("FICHER JSON : ")
13 print (personnes_ISS_JSON)
14 print (" ")
15

```

```
16 # pprint permet juste un affichage plus aéré et mieux organisé dans la console
17 print ("AFFICHAGE ORGANISE DES DONNEES : ")
18 pprint.pprint(personnes_ISS_JSON)
19 print (" ")
20
21 # Affichage du nombre de personnes dans la station internationale en ce moment
22 print ("nombre de personnes dans l'ISS : {}".format(personnes_ISS_JSON['number']))
23
24 # Affichage du nom de chaque personne
25 for chacun in personnes_ISS_JSON['people'] :
26     print ("nom : {}".format(chacun['name']))
```

Testez ce code depuis Thonny pour vérifier les informations affichées dans la console.

Si vous remontez dans l’affichage, vous verrez l’affichage des données brutes depuis le fichier ‘**astros.json**’ puis la mise en forme par le programme Python qui extrait uniquement les données qui nous intéressent.

Les commentaires dans le code vont vous aider à comprendre le fonctionnement de ce code, mais je complète avec quelques informations supplémentaires ci-dessous :

- Ligne 2 : le module **requests** est un module qui permet de récupérer toute sorte de contenu sur le web. Nous l’avons déjà utilisé lors de la séance précédente.
- Ligne 3 : le module **pprint** est un simple module de mise en forme pour l’affichage d’objet de type dictionnaire par exemple.
- Ligne 13 : le résultat de cet affichage est le contenu brut du fichier JSON sans aucune mise en forme.
- Ligne 18 : les données JSON ne sont pas modifiées, il s’agit juste d’une fonction d’affichage pour obtenir un résultat plus lisible pour nous.
- Ligne 22 : Vous retrouvez l’accès au nombre de personnes présentes dans l’ISS comme on l’a évoqué dans la page précédente.
- Ligne 25 : L’affichage de la ligne 18 nous a permis de voir que la clef ‘people’ était associée à une liste (au sens propre du terme python) contenant un dictionnaire pour chaque personne présente. Cette boucle ‘for’ permet de récupérer chaque dictionnaire de cette liste (dans l’objet ‘chacun’).
- Ligne 26 permet d’afficher, pour le dictionnaire correspondant à la personne pointée par la boucle ‘for’, son nom grâce à la clef ‘name’.

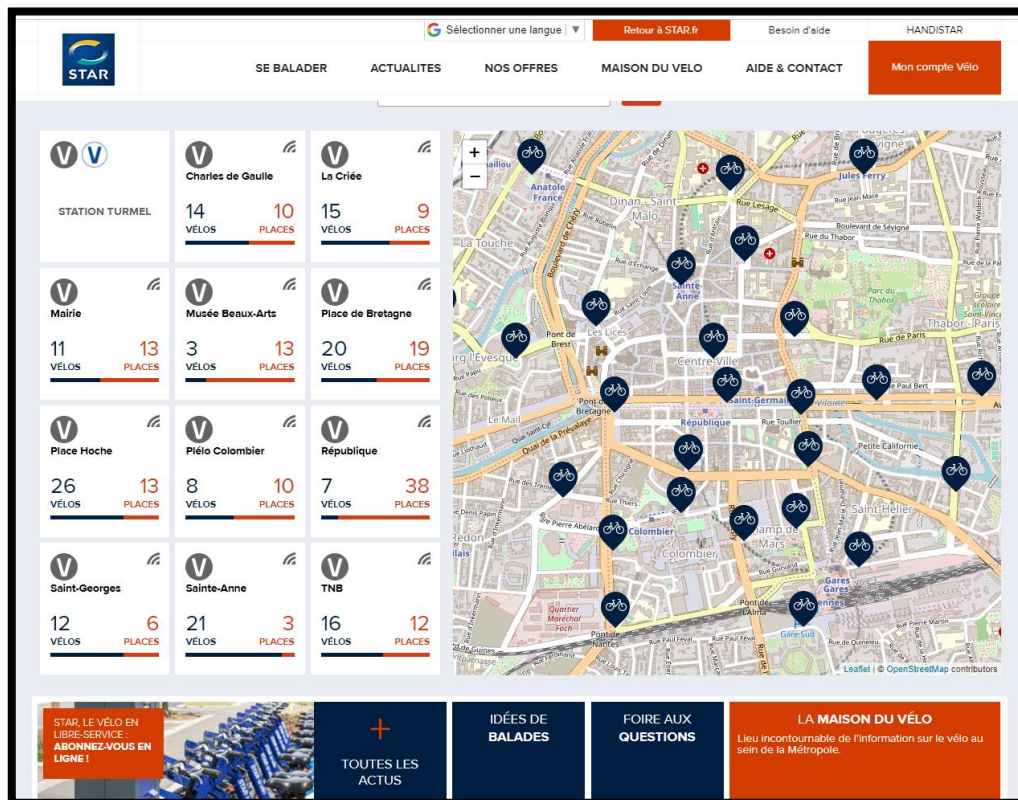
Prenez le temps de bien comprendre cet exemple car vous allez devoir utiliser une démarche similaire pour l’exercice qui suit.

EXERCICE 1 : ‘Ex01.py’

Nous allons donc exploiter ce que nous venons de voir pour obtenir des informations depuis le réseau STAR de Rennes. On va plus particulièrement s’intéresser à obtenir pour chaque station de vélo en libre-service, le nombre de vélos restant en temps réel.



Avec ce que l'on a vu dans la séance de TP précédente, on pourrait simplement extraire de la page web <https://www.star.fr/le-velo/> avec l'aide du module 'BeautifulSoup' :



Toutefois, on ne va procéder ainsi pour deux raisons :

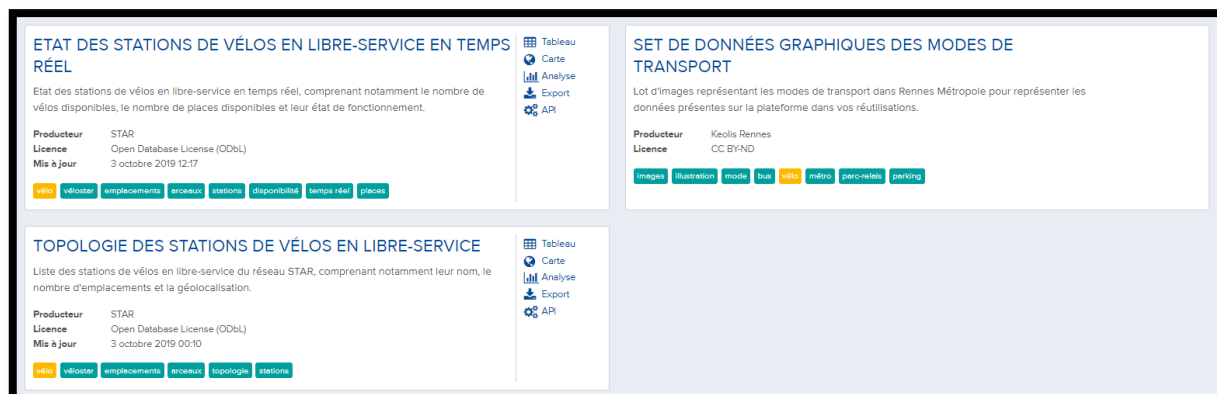
1. La première est incontournable : toutes les stations n'apparaissent pas sur la page d'accueil. Il faut dézoomer sur la carte et cliquer sur une autre station pour voir son affichage.
2. La seconde est que la STAR met à disposition ces informations via leur site : <https://data.explore.star.fr/page/home/>

Pour travailler comme dans le tutoriel, il va vous falloir l'adresse exacte du fichier JSON qui nous fournira les vélos disponibles pour chacune des stations du parc.

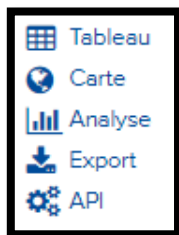
Pour cela, rendez vous dans l'onglet 'Explorer' du site : <https://data.explore.star.fr/explore/?sort=title>

Cela vous permet déjà de voir, en parcourant la page, le nombre d'informations fournies par la STAR aux développeurs.

Filtrez l'affichage en ne sélectionnant que le mot clef 'vélo' sur la gauche et vous devriez obtenir les résultats suivants :



La zone '**ETAT DES STATIONS DE VELOS EN LIBRE-SERVICE EN TEMPS REEL**' va nous fournir la réponse à notre recherche. Les liens accompagnés d'une icône vont nous aider :



Un clic sur '**Tableau**' nous affiche la liste et les informations attendues au sein du navigateur (remarque : ici, il pourrait être plus judicieux d'utiliser 'BeautifulSoup' pour extraire des infos car toutes les stations sont présentes sur cette page. Il suffirait d'utiliser l'adresse suivante : <https://data.explore.star.fr/explore/dataset/vls-stations-etat-tr/table/>).

'**Carte**' permet d'afficher une carte dynamique en ligne avec toutes les stations vélo du réseau.

'**Analyse**' permet de faire en ligne des analyses sur l'utilisation des vélos.

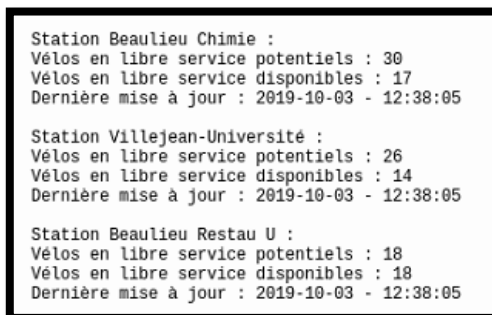
'**API**' nécessite une identification auprès de la STAR et permet de personnaliser son jeu de données pour l'exploiter à distance.

'**Export**' est celui qui va nous servir dans cet exercice. Il permet de récupérer un fichier dans différents formats contenant toutes les informations que contenait la partie '**Tableau**'.



Vous pouvez éventuellement télécharger le fichier pour l'afficher dans un éditeur de texte, mais ce qui va nous être le plus utile c'est de récupérer le lien de ce fichier.

Maintenant que vous avez le lien vers le fichier JSON qui contient, entre autres, les données des vélos disponibles pour chaque station du réseau STAR de Rennes, vous allez pouvoir appliquer ce que vous avez vu au tutoriel n°1 pour afficher dans la console, le nombre de vélos disponibles pour chaque station du réseau :



Remarque : Pour cet exercice, vous n'êtes pas obligé d'afficher l'heure de la dernière MAJ.

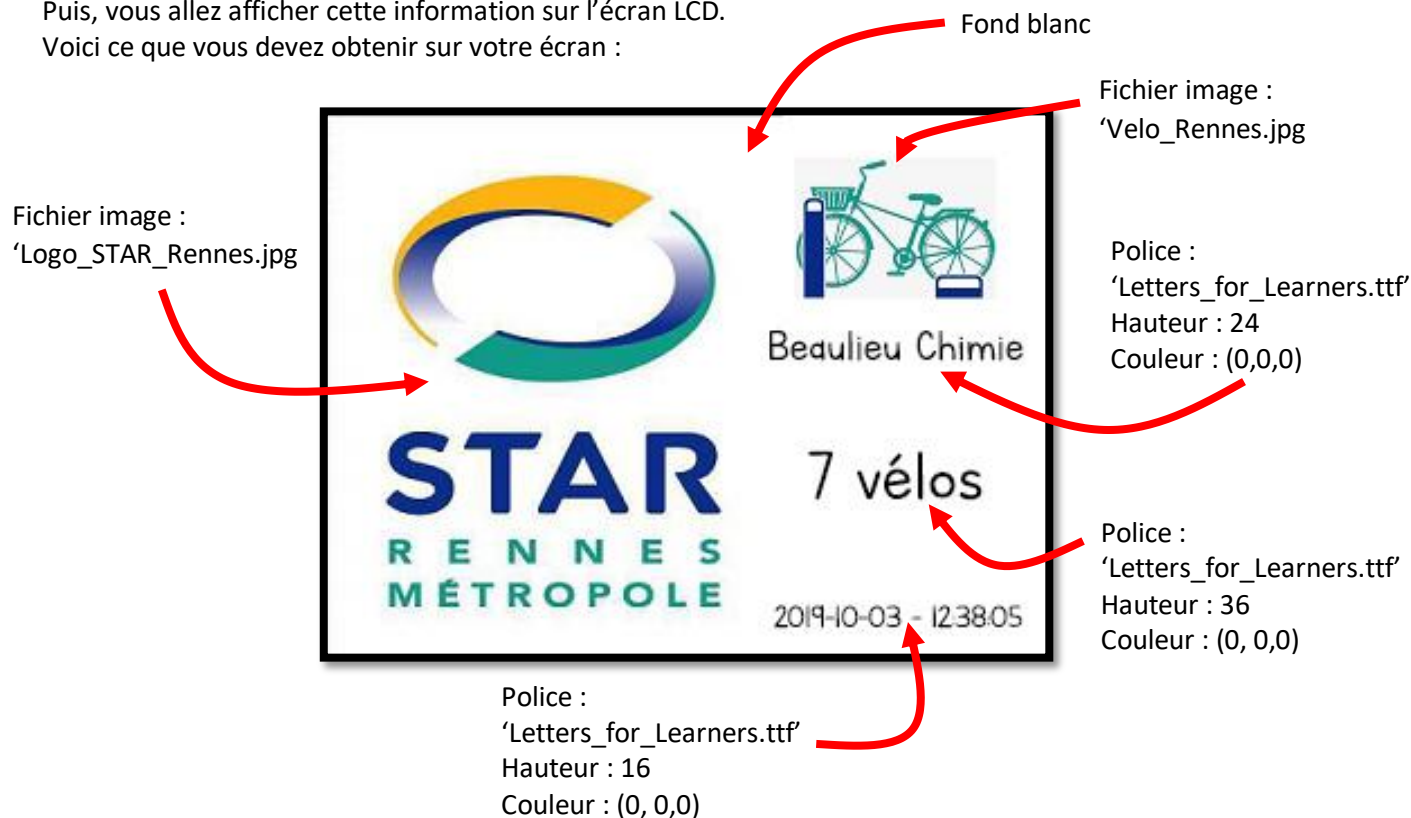
Faites contrôler le bon fonctionnement de votre programme par un enseignant.

EXERCICE 2 : 'Ex02.py'

Dans cet exercice, vous allez modifier le code précédent en demandant à l'utilisateur la station dont il veut connaître le nombre de vélos disponibles.

Puis, vous allez afficher cette information sur l'écran LCD.

Voici ce que vous devez obtenir sur votre écran :



N'oubliez pas qu'il faut que le fichier '**lib_tft24T.py**' soit dans votre dossier TP7

Veuillez faire contrôler le bon fonctionnement de votre application par un enseignant.

RECUPÉRATION DE DONNÉES ISSUE DU WEB VIA UNE API

TUTO 2 : 'Tuto02.py'

On va s'intéresser à la solution la plus conviviale pour accéder à des données sur le web. Il s'agit d'utiliser une API avec sa librairie Python associée.

L'API que nous allons explorer est celle liée à la recherche d'images dans le moteur Google :



Pour cela, on va installer un nouveau module Python via le terminal :

\$ sudo pip3 install google_images_download

Cette librairie est une librairie de très haut niveau permettant de récupérer physiquement sur la carte SD des images via Google à partir d'une recherche par mots clefs.

Dans le cadre de ce module, une partie du travail a été fait en amont et vous n'avez qu'à suivre les instructions des textes de TP pour aboutir aux résultats attendus. Dans d'autres circonstances, comme en stage par exemple, il serait inconcevable de ne pas lire la documentation en ligne de l'API.

Pour information, sa page officielle est : <https://github.com/hardikvasa/google-images-download>

Et sa documentation est à la page : <https://google-images-download.readthedocs.io/en/latest/index.html>

Ouvrez le fichier 'Tuto02.py' dans Thonny :

```
1 # Importation des librairies
2 import os
3 # sudo pip3 install google_images_download
4 # http://www.github.com/hardikvasa/google-images-download pour un lien vers la doc de l'API
5 from google_images_download import google_images_download
6
7 # Nombre d'images à télécharger
8 NB_IMG = 15
9
10 # Creation du dossier 'google_search' s'il n'existe pas
11 if not os.path.exists("google_search") :
12     os.makedirs("google_search")
13
14 # Effacement de tous les fichiers du dossier 'google_search'
15 try :
16     fichiers_presentes = [ f for f in os.listdir("google_search")]
17     for f in fichiers_presentes:
18         os.remove(os.path.join("google_search", f))
19 except FileNotFoundError :
20     print("...")
21
22 # Instantiation de l'objet 'google'
23 reponse = google_images_download.googleimagesdownload()
24
25 # Demande des mots clefs à l'utilisateur
26 mots_clefs = input("Quels mots clefs souhaitez-vous pour votre recherche ? ")
27
28 # Récupération des images du résultat de la recherche
29 absolute_image_paths = reponse.download({"keywords":mots_clefs,"limit":NB_IMG,"format":"jpg","output_directory":"google_search","no_directory":'1'})
30
31 # renommer les images en changeant le nom web en index uniquement (ex: '8.jpg')
32 i=0
33 for f in os.listdir("google_search") :
34     os.rename("google_search/"+f,"google_search/"+str(i)+".jpg")
35     i+=1
```

Comme d'habitude, on va compléter les commentaires par quelques informations :

- Ligne 8 : le nombre d'images total à télécharger. Vous pouvez réduire ce nombre, mais ne l'augmentez pas sous peine de voir la durée d'exécution de votre code s'allonger inutilement.
- Ligne 11-12 : on utilise ici le module 'os' pour vérifier la présence d'un dossier et pour la création d'un dossier.
- Lignes 15-20 : Effacement du contenu du dossier créé pour assurer le fonctionnement du programme à chaque nouveau lancement.
- Ligne 26 : l'utilisateur a la possibilité de fournir plusieurs mots clefs séparés par un espace et même d'utiliser les guillemets ou les mots clefs 'OR', 'AND', ... comme dans la barre de recherche du site web de Google.
- Ligne 29 : On utilise ici l'API avec comme paramètres les mots clefs de l'utilisateur, le nombre d'images à télécharger, le format des images, le répertoire de sauvegarde,... Le nombre de paramètres possibles est impressionnant. Vous les retrouverez tous à l'adresse suivante : <https://google-images-download.readthedocs.io/en/latest/arguments.html>
- Lignes 32-35 : on renomme les fichiers '0.jpg', '1.jpg', '2.jpg', ... pour faciliter leur utilisation dans les prochains exercices. Comme à chaque lancement du programme, les fichiers image seront renommés de la même façon, on comprend plus facilement pourquoi on efface le contenu du dossier aux lignes 12-20

Testez ce code depuis Thonny et vérifiez les images téléchargées dans le dossier 'google_search'.

EXERCICE 3 : 'Ex03.py'

Vous allez partir de la base du code fourni dans le tutoriel précédent pour y ajouter l'affichage sur le LCD.

Dans une boucle infinie, vous allez lire et afficher une image à la fois sur l'écran LCD :



Informations utiles :

- Vous laisserez une pause de 2 secondes entre chaque image.
- Votre boucle doit utiliser comme valeur maximale la variable '**NB_IMG**' du tutoriel n°2
- Vous allez vous heurter à une difficulté importante qui concerne l'apparence et la taille des images. En effet sur le web, la résolution des images est variable. On passe de petites icônes en 16x16 à des images 4k ou des textures ou des cartes en très haute résolution. Certaines images sont en mode portrait, d'autres en mode paysage. Comment gérer cela lorsque l'on ne dispose pour son affichage que d'un écran de 320x240 ?? Deux solutions se présentent à vous :
 - Soit vous ajouter dans les paramètres de recherche les dimensions des images à rechercher ('**exact_size**'), ainsi que son aspect ('**aspect_ratio**'). Pour cela, il faut vous aider de la page : <https://google-images-download.readthedocs.io/en/latest/arguments.html>
 - Soit, si vous ne voulez pas réduire votre champ de recherche, vous devez réduire leur taille via votre code Python. Pour cela, on doit utiliser la méthode '**thumbnail**' de la librairie PIL. Voici le code à implanter juste après avoir téléchargé et renommé les images :

```
# transformation des images pour les mettre au format de l'écran LCD
for i in range(NB_IMG) :
    resultat = Image.open('google_search/' + str(i) + '.jpg')
    resultat = resultat.rotate(-90, expand = True)
    if resultat.size[0] >= resultat.size[1] :
        resultat.thumbnail((220,220))
    else :
        resultat.thumbnail((300,300))
    resultat.save('google_search/' + str(i) + '.jpg')
```

'thumbnail' va adapter la plus grande dimension de l'image
aux paramètres qu'on lui fournit.

Exemples :

Image paysage 1024x768 (ratio 1,33) → 300x225 (ratio 1,33)

Image portrait 768x1024 (ratio 0,75) → 165x220 (ratio 0.75)

Veuillez faire contrôler le bon fonctionnement de votre application par un enseignant.

TUTO 3 : 'Tuto03.py'

Sur la carte d'extension, il n'y a pas que l'écran LCD, il y a notamment le capteur de gestes APDS-9960. C'est ce capteur que l'on va mettre en œuvre dans ce tutoriel



Ce capteur est basé sur une détection infrarouge.
Une LED infrarouge émet un rayonnement et quatre photodiodes reçoivent ou non un rayonnement réfléchi par un obstacle.
Tout le traitement des signaux et leur interprétation sont réalisés par le capteur.
Nous avons donc juste à l'interroger (protocole I2C) pour savoir ce qu'il a détecté !!

Bien évidemment, tout comme pour le LCD ou tout autre composant, python ne dispose pas de librairie native pour piloter notre capteur de gestes. N'oubliez pas que Python n'a pas été développé et n'est toujours pas développé dans le but de piloter du hardware. Dès que l'on a du matériel à piloter, il faudra donc toujours installer une librairie dédiée ou l'écrire soi-même !!

Pour le capteur de gestes APDS-9960, plusieurs librairies sont disponibles sur le web. Là encore, j'ai fait la sélection en amont pour faciliter le travail en TP, mais normalement, c'est à chacun de voir si la librairie fonctionne sous Python 2 ou 3, si elle répond aux besoins que l'on s'est fixés, ...

J'ai choisi ici une librairie qui ne nécessite pas d'installation, rendant ainsi le code transportable. Elle est disponible à l'adresse : <https://github.com/buxtonpaul/APDS-9960-PY>

Pour la récupérer, vous devez la cloner via le terminal :

\$ git clone <https://github.com/buxtonpaul/APDS-9960-PY>

Dans le dossier ainsi créé, récupérez les fichiers '**apds9960.py**' et '**apds9960_constants.py**' et copiez-les dans le répertoire du TP7.

On ne va pas du tout s'intéresser aux contenus de ces fichiers mais plutôt les voir comme des boîtes noires qui vont nous fournir des informations sur les gestes réalisés au-dessus du module.

Toutefois, la librairie n'est pas documentée sur les méthodes et attributs disponibles pour cette classe d'objet.

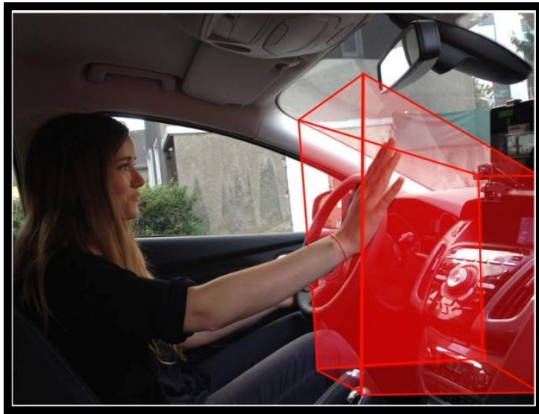
Vous allez donc simplement ouvrir le fichier '**Tuto03.py**' dans Thonny et l'exécuter.

Prenez le temps de lire les commentaires pour bien comprendre le fonctionnement du code car vous allez devoir l'exploiter dans l'exercice suivant.

EXERCICE 4 : 'Ex04.py'

Maintenant que l'on a pu tester le module 'APDS-9960' et vérifier qu'il détecte bien nos gestes, on va l'intégrer dans notre recherche d'images.

La détection par gestes se développe de plus en plus dans de nombreux domaines. La command in-air dans les véhicules ou encore la gestion améliorée des gestes dans la version Q d'Android sortie en septembre 2019 :



Le but de cet exercice est finalement de transformer le code de l'exercice n°3 : on ne va plus faire défiler en boucle les images contenues dans le dossier 'google_search' mais on va les faire défiler en détectant des gestes :

- Balayage gauche : affichage de l'image précédente
- Balayage droit : affichage de l'image suivante

Informations utiles :

- Vous limiterez la fonction d'interruption du capteur APDS-9960 ('**Lecture_gestes()**') aux deux balayages gauche et droite.
- Vous devez utiliser maintenant à la place de votre boucle d'affichage une simple variable d'index (compris entre 0 et NB_IMG) qui augmentera ou décrémentera d'une unité à chaque détection de gestes.
- Dans cette fonction d'interruption, vous devez donc selon le geste détecté, incrémenter ou décrémenter votre variable d'index et faire en sorte qu'elle n'atteigne pas NB_IMG ou qu'elle devienne négative.
- Débrouillez vous également pour que l'écran ne s'actualise que lorsqu'un geste a été détecté.

Faites vérifier le fonctionnement de votre code par un enseignant.

TUTO 4 : 'Tuto04.py'

Le potentiel de python est immense, en particulier via ses modules de traitement d'images. On va exploiter ici un module qui va nous permettre d'extraire la couleur dominante d'une image.

Ce module est présent à l'adresse : <https://github.com/pixelogik/ColorCube>

Rendez vous sur cette page pour voir un peu le principe de la détection.

Un fichier va nous être utile, c'est le fichier 'ColorCube.py' mais il est déjà dans votre dossier TP7 que vous avez cloné depuis Github en début de séance.

Ce module permet de fournir, à partir d'une image quelconque, les couleurs dominantes d'une image par ordre d'importance.

Pour nous, on va se contenter de récupérer la première d'entre-elles. Ouvrez le fichier 'Tuto04.py' dans Thonny :

```
1  # Importation des librairies utiles
2  from ColorCube import ColorCube
3  from PIL import Image
4
5  # Instanciation de l'objet color cube, en précisant d'éliminer les couleurs trop proches du blanc
6  cc = ColorCube(avoid_color=[255, 255, 255])
7
8  # Chargement d'une image et redimensionnement à la taille (100,100) pour
9  # rendre le temps de traitement plus rapide
10 # En diminuant la taille, on intensifie aussi la perception des couleurs dominantes
11 image = Image.open("images/lego-yellow-brick.jpg").resize((100,100))
12
13 # Traitement de l'image et extraction des couleurs
14 colors = cc.get_colors(image)
15
16 # Affichage des trois premières couleurs dominantes
17 print ("couleur dominante 1 : {}".format(colors[0][0]))
18 print ("couleur dominante 2 : {}".format(colors[0][1]))
19 print ("couleur dominante 3 : {}".format(colors[0][2]))
```

Exécuter ce code et relevez la première couleur obtenue. Vous pouvez vous rendre à cette adresse pour voir à quoi correspond le résultat fourni : <https://cssgenerator.org/rgba-and-hex-color-generator.html>

Vous pouvez également changer le nom du fichier image de la ligne 11 en regardant les fichiers disponibles dans le sous-dossier 'images' de votre dossier TP7.

EXERCICE : 'Ex05.py'

On va ici modifier légèrement le code de l'exercice n°4 en y intégrant ce que l'on a vu dans le tutoriel précédent. L'objectif est d'extraire la couleur dominante de l'image google affichée à l'écran et de remplir l'arrière-plan du LCD avec cette couleur.

En effet, lorsque vous affichez les images issues de votre recherche google, elles n'occupent pas tout l'écran la plupart du temps car elles n'ont pas le même ratio que l'écran (d'autant plus qu'en prévision, à l'exercice 3, on les a redimensionnées en 300x220 max).

Modifiez donc légèrement le code 'Ex04.py' pour y intégrer l'affichage de cette couleur dominante en arrière-plan. Pour cela, vous devez modifier la couleur au sein de la commande '**TFT.clear((0,0,0))**' qui par défaut efface l'écran en le remplissant de la couleur noire.

Attention, pensez bien dans votre boucle principale à extraire les couleurs de l'image à chaque nouvelle image affichée.

Voici à quoi devrait ressembler votre écran en reprenant l'image de l'exercice 3 :



Remarque : si vous le désirez, vous pouvez améliorer l'aspect visuel en ajoutant un fin bord noir autour de votre image. Il suffit pour cela d'avoir récupéré les dimensions de l'image affichée (via l'attribut '**.size**'). Il vous suffit alors de tracer un rectangle noir légèrement plus grand (2 pixels) que l'image (via '**zone_ecran.rectangle()**') avant de mettre l'image à proprement parlé (via '**zone_ecran.pasteimage()**') :



Faites vérifier le fonctionnement de votre code par un enseignant.

TUTO 5 : 'Tuto05.py'

Sur la carte d'extension, il y a également deux LEDs qui sont cachées en dessous. Ce sont des LEDs intelligentes (comprenez par là communicantes) appelées NEOPIXEL. Leur référence est WS2812.

On ne va pas chercher à comprendre le fonctionnement de ces LEDs, mais simplement d'utiliser une librairie qui va nous permettre de les piloter.

Pour cela, installez la librairie suivante via un terminal :

```
$ sudo pip3 install rpi_ws281x adafruit-circuitpython-neopixel
```

Malheureusement, ce type de librairie utilise la broche PWM de la puce de la Raspberry et cette dernière ne peut être employée qu'en mode administrateur.

Fermez donc 'Thonny' s'il est encore ouvert et dans un terminal, tapez l'instruction suivante :

```
$ sudo thonny &
```

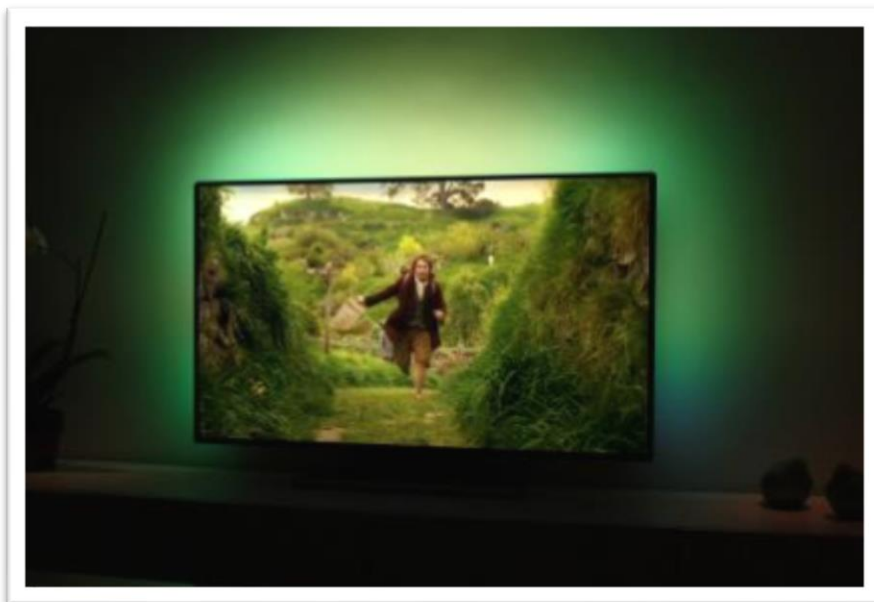
Dès lors, vous pouvez ouvrir le fichier 'Tuto05.py' et le tester.

Prenez un peu de temps pour le comprendre (il est très simple !!) et cela vous sera nécessaire pour le dernier exercice.

EXERCICE 6 : 'Ex06.py'

L'objectif de ce dernier exercice est de compléter à nouveau le code de l'exercice n°5 de manière que les LEDs NEOPIXEL prennent la couleur dominante de l'image affichée à l'écran.

On obtiendra ainsi un effet visuel se rapprochant de ce que proposent les téléviseurs Philips avec leur AMBILIGHT :



Vous ne retirez rien du programme de l'exercice n°5, vous devez juste ajouter certaines lignes du tutoriel précédent pour obtenir l'effet visuel voulu : le fond de l'écran LCD et les LEDs doivent avoir la même couleur et cette dernière doit correspondre à la couleur dominante de votre image.

Voici l'effet attendu :



Faites vérifier le fonctionnement de votre code par un enseignant.

DEMO : 'demo_api.py'

Un dernier exemple sous forme de démo pour vous montrer la facilité apportée par l'utilisation d'API dédiée en python. Ici, c'est l'API pour Wikipedia qui va être utilisée.

Avant cela, il faut l'installer via un terminal :

```
$ sudo pip3 install wikipedia
```

Si vous n'avez pas fait le dernier exercice du TP n°1, il vous faudra aussi les deux librairies suivantes :

```
$ sudo pip3 install gTTS
```

```
$ sudo apt-get install mpg321
```

Branchez un haut-parleur comme celui proposé dans le TP n°1, ouvrez le fichier '**demo_api.py**' et testez-le en fournissant un mot-clef.

