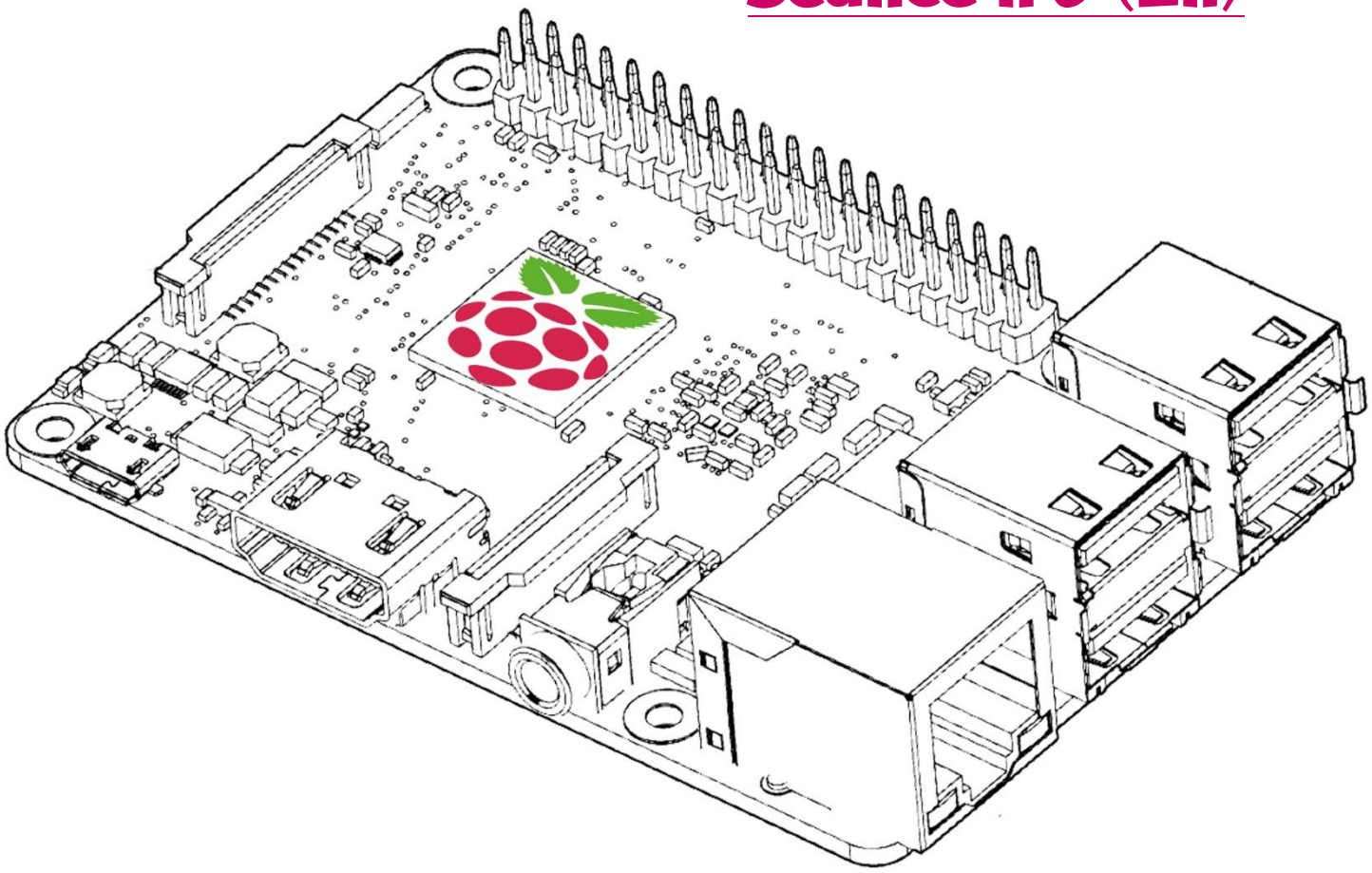
AT31 - Module IPE

IUT RENNES

Séance n°6 (2h)



Utilisation d'APIs
partie 1

Michaël BOTTIN

Objectifs de la séance :

- Découvrir l'utilisation d'un écran LCD couleurs
- Récupération de données sur le web via différentes méthodes (web scraping, CSV et RSS)

Récupération des fichiers utiles pour cette séance :

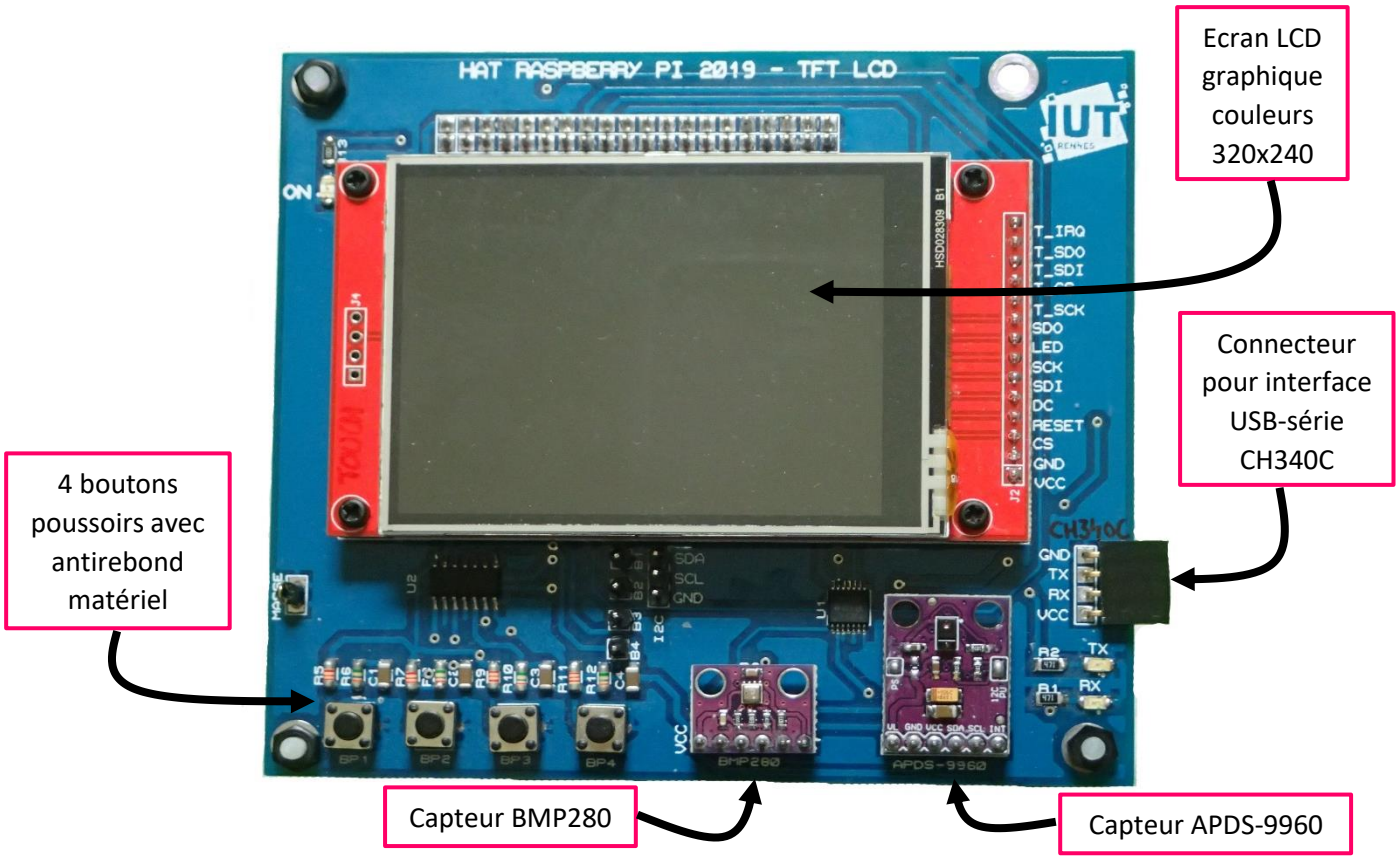
Comme pour le TP précédent, il vous faut récupérer les fichiers utiles pour l'écriture de vos programmes.
Vous allez les récupérer depuis **Github**.

Pour cela, tapez la commande suivante dans le terminal :
$ git clone https://github.com/MbottinIUT/IUT_IPE_TP6.git
[remarque : il n'y a pas d'espace dans ce lien, juste des '_']

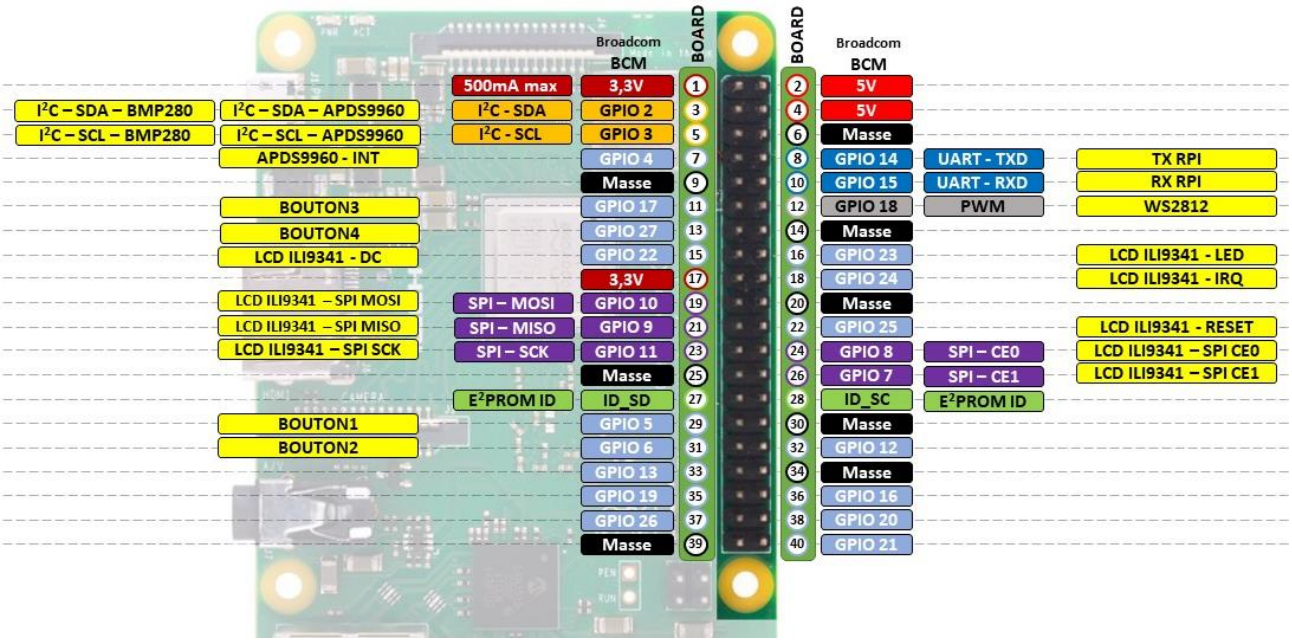
Une fois la tâche effectuée, vous devriez retrouver, en ouvrant un explorateur de fichiers, un dossier '**IUT_IPE_TP6**' sous '**/home/pi**'.

Présentation de la carte que l'on va utiliser :

A partir d'aujourd'hui et jusqu'à la fin du module, nous allons utiliser une nouvelle carte d'extension.
Celle-ci, tout comme la carte 'vumètre' a également été conçue à l'IUT (sauf la réalisation du circuit imprimé qui a été sous-traitée).
Voyons un peu à quoi ressemble cette nouvelle carte :

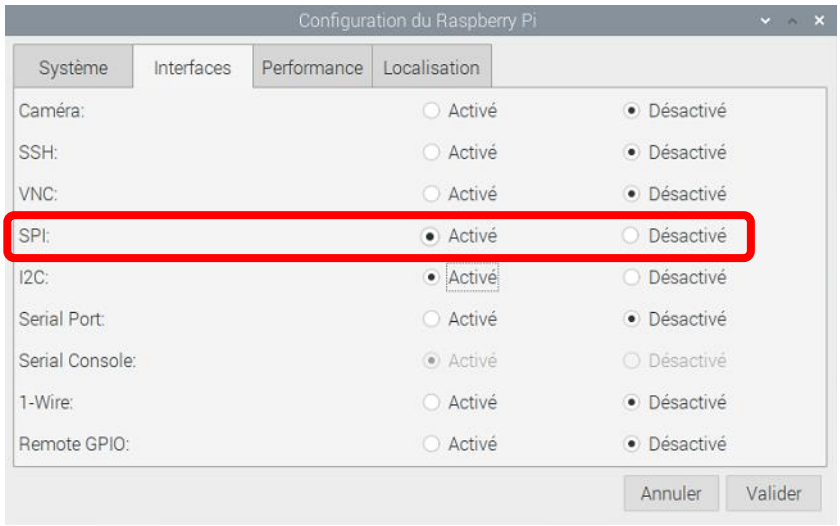


Comme pour la carte 'vumètre', si l'on veut utiliser ces différents éléments, il faudra dans la programmation indiquer le plus souvent sur quelle(s) broche(s) ils sont connectés. Il est donc fondamental d'avoir un schéma de câblage. J'aurais pu vous fournir le schéma structurel sous Proteus, mais je me suis contenté ici d'un schéma de brochage qui est bien suffisant dans notre cas :



On donnera quelques détails complémentaires lorsque nous utiliserons chacun des composants de la carte.

Toutefois, on peut tout de même remarquer sur le schéma précédent que les broches du bus I²C sont utilisées, tout comme celles du bus SPI.
Or, jusqu'à maintenant, nous n'avions eu besoin que du bus I²C, il est donc important d'activer le bus SPI dans les interfaces de la Raspberry Pi.
Pour cela, il vous faut ouvrir la fenêtre des préférences de la Raspberry via le menu 'démarrer' puis aller sur l'onglet '**interfaces**' pour valider le '**SPI**' :



Vérifiez bien que l'I²C est toujours activé puisque nous en aurons également besoin.

Voilà, nous allons pouvoir démarrer nos exercices de programmation en commençant par la découverte de l'écran LCD.

TUTO 1 : 'Tuto01.py'

Dans ce premier tutoriel, nous allons mettre en œuvre l'écran LCD en y affichant du texte et des images.

La Raspberry Pi accepte un nombre très important d'écrans LCD de taille et d'interfaces diverses. Et un écran est équipé la plupart du temps d'un composant qui le pilote (driver). Pour ce LCD, le driver a comme référence : ILI9341.
Pour pouvoir afficher des informations sur l'écran, il faudrait donc normalement étudier la documentation du composant ILI9341 pour savoir comment communiquer avec lui. Heureusement, la plupart du temps, d'autres l'ont fait avant nous !!
Il faut donc trouver une librairie en Python 3 pour piloter cet écran.
L'une d'elles est disponible à l'adresse suivante : https://github.com/BLavery/lib_tft24T
Mais vous n'avez pas besoin de vous rendre à cette adresse.
Le fichier de librairie qui permet de commander l'écran se nomme '**lib_tft24T.py**'. Vous le trouverez dans le dossier que vous avez cloné de Github en page 2. Vérifiez qu'il est bien dans le dossier du TP6 afin de pouvoir l'utiliser dans votre propre code.

Dans Thonny, vous allez ouvrir le fichier '**Tuto01.py**' qui correspond à toute l'initialisation :

```
# Importation des librairies propres aux images
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

# Importation des libraries pour la gestion de l'écran
from lib_tft24T import TFT24T
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
import spidev

# Déclaration des numéros de broches utiles pour la gestion de l'écran
DC =
RST =
LED =

# Instanciation de l'objet LCD
TFT = TFT24T(spidev.SpiDev(), GPIO)

# Initialisation de l'écran
TFT.initLCD(DC, RST, LED)

# Création d'un buffer représentant la zone d'affichage de l'écran
zone_ecran = TFT.draw()
```

Pour la déclaration des broches du LCD, vous devez juste fournir les numéros de broches pour DC, RST (RESET) & LED conformément au schéma de câblage de la page précédente (utiliser les numéros GPIO).

Ces lignes seront communes à tous les exercices que nous allons réaliser dans ce TP et dans le suivant.

Pour information :

- La librairie '**PIL**' (Pillow) est une librairie qui permet de manipuler des images en Python.
- La librairie '**Rpi.GPIO**' permet de gérer les broches de la Raspberry. On a utilisé une librairie plus récente dans le TP2 qui remplissait ce rôle : 'gpiozero'. Toutefois, 'Rpi.GPIO' est nécessaire ici car la librairie qui gère l'écran LCD l'utilise.
- La librairie '**lib_tft24T**' permet de gérer l'écran LCD.
- La librairie '**spidev**' permet de gérer la communication sur le bus SPI entre la Raspberry et l'écran LCD.

Dû à cette imbrication dans les librairies, il est toujours très important que la librairie que l'on veut utiliser soit parfaitement documentée.

A la fin de cette initialisation, vous disposez de **deux objets** qu'il faut bien distinguer :

- Un objet '**TFT**' qui représente physiquement la zone d'affichage de l'écran. Cette classe n'a pas d'attribut mais dispose de plusieurs méthodes comme :

TFT
Pas d'attribut
initLCD(DC, RST, LED) clear(color=(r,g,b)) draw()

- Un objet '**zone_ecran**' qui représente un écran virtuel sur lequel on va pouvoir ajouter tout ce que l'on désire afficher sur l'écran : texte, graphique, image, … Cet objet hérite de tous les attributs et les méthodes de la classe '**PIL**'. Il est impossible de tous les énumérer ici, tant les fonctionnalités sont nombreuses. Pour vous en convaincre, vous pouvez jeter un coup d'œil à sa documentation en ligne : https://pillow.readthedocs.io/en/stable/

Cela peut vous sembler obscur pour le moment, mais vous allez vite comprendre en écrivant quelques lignes de code supplémentaires.
Ce qu'il faut juste retenir, c'est que l'on 'dessine' dans '**zone_ecran**' tout ce dont on a besoin et au moment où l'on désire l'afficher, on appelle la méthode '**TFT.display()**'. Si vous ne l'appelez pas, rien ne s'affichera à l'écran !!

A la suite de l'initialisation du fichier '**Tuto01.py**', nous allons effacer l'écran. Cela se fait par la commande :
TFT.clear((0,0,0))
Ici, l'écran sera noir puisque les composantes (R,G,B) sont toutes à 0.

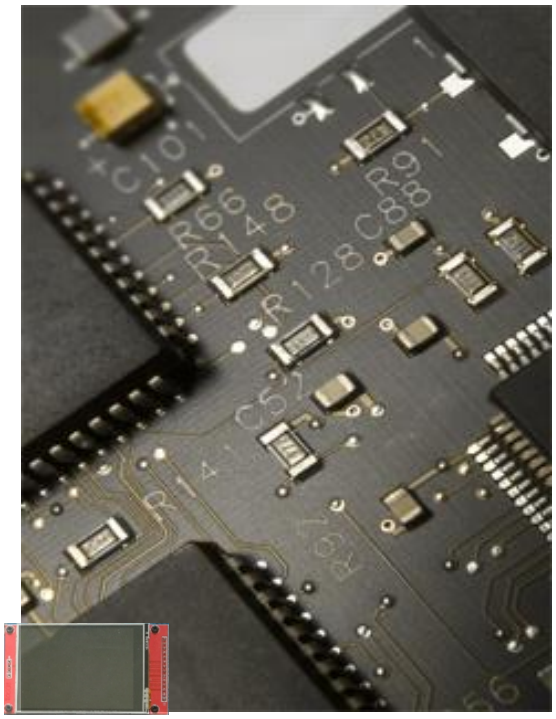
On va maintenant chercher à afficher une image. Pour cela, ajoutez à la suite la commande :
zone_ecran.pasteimage('images/background.jpg', (0,0))

(0,0) représente la position (X,Y) du coin gauche de l'image sur l'écran

Ensuite, ajoutez simplement :
TFT.display()
Pour valider l'affichage.
Observez le résultat de l'exécution de ce code et parallèlement, ouvrez ce fichier image via l'explorateur pour comparer.
Qu'observez-vous ? L'expliquez-vous sachant que la résolution de notre écran est 320x240 pixels ?
Remarquez également l'orientation de l'image pour repérer où est l'origine des coordonnées.

On va corriger ceci en 'préparant' dans notre code notre image avant son affichage.

Voici l'image complète et en bas à gauche la portion de l'image que l'on a obtenue sur l'écran :

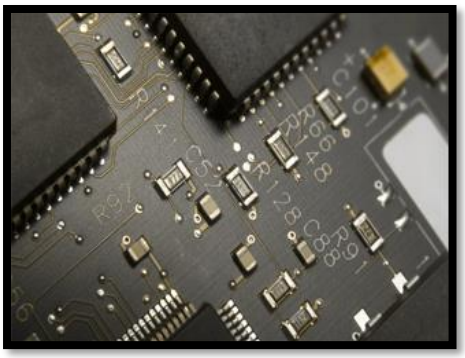


Il va donc falloir tourner l'image de 90° mais également la redimensionner pour la visualiser en entier sur l'écran.
On va donc supprimer la ligne **zone_ecran.pasteimage('images/background.jpg', (0,0))**

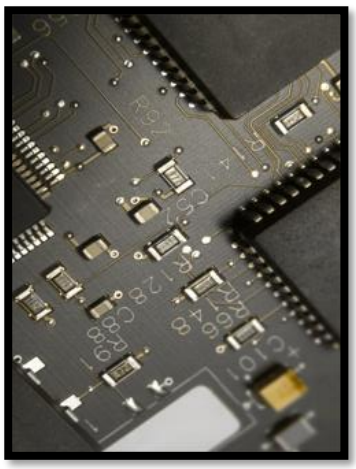
A la place, ajoutez les lignes suivantes :

1. **fond = Image.open('images/background.jpg')**
2. **fond = fond.rotate(-90, expand=True)**
3. **fond = fond.resize((240,320))**
4. **fond.save('fond.jpg')**

Instruction 1



Instruction 2



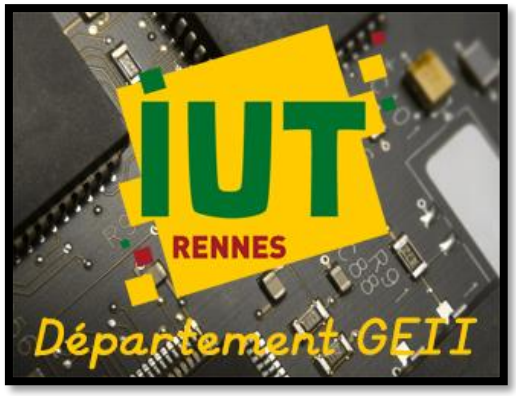
Instruction 3

La nouvelle image tournée de 90° et redimensionnée est sauvegardée dans votre dossier '**IUT_IPE_TP6**' sous le nom '**fond.jpg**'.

On peut donc maintenant ajouter à la suite l'instruction suivante :
zone_ecran.pasteimage('fond.jpg', (0,0))
Vérifiez que le résultat correspond à nos attentes.

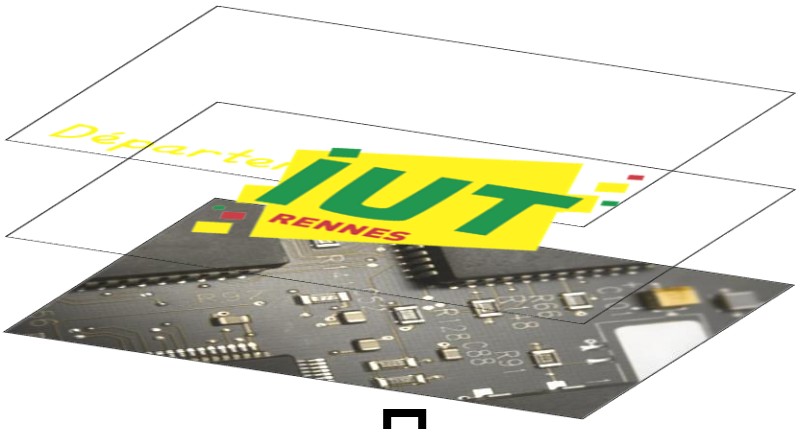
Remarque : j'aurais pu préparer en amont mes images dans un logiciel de photo comme GIMP ou PHOTOSHOP afin qu'elles aient la bonne orientation et la bonne taille, mais volontairement je ne l'ai pas fait. Même si cela vous semble un peu laborieux, cela permet tout de même de montrer toute la puissance d'un langage comme Python.

On va continuer notre tutoriel en cherchant à afficher sur l'écran la composition suivante :



On a déjà affiché le fond.
On va donc chercher à afficher le logo de l'IUT (avec une couche alpha → transparence) puis du texte dans une police définie.
Il faut imaginer que chaque élément sera placé sur un 'calque' et que l'instruction '**TFT.display()**' permet d'aplatir l'image et de l'afficher. Bien sûr l'ordre des calques et donc des instructions liées à la '**zone_ecran**' est primordial si l'on ne veut pas qu'une portion d'un calque en cache un autre !!

Travail sur '**zone_ecran**' :

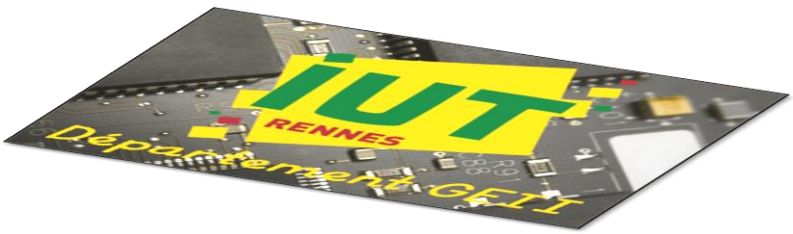


zone_ecran.textrotated()

zone_ecran.pasteimage()

zone_ecran.pasteimage()

Travail sur '**TFT**' :



TFT.display()

Pour effectuer tout cela, vous allez ajouter les lignes de code suivantes avant le '**TFT.display()**' :

```
# Modification de l'image du logo de l'IUT et sauvegarde locale
logo = Image.open('images/logo-iut-rennes.png')
logo = logo.rotate(-90, expand=True)
logo.save('logo.png')

# Fusion des images logo et fond pour conserver la transparence
img1 = Image.open('fond.jpg')
img2 = Image.open('logo.png')
img1.paste(img2, (0,0), img2)
img1.save('composition.jpg')
zone_ecran.pasteimage('composition.jpg', (0,0))

# Affichage de texte dans la police choisie
zone_ecran.rectangle(((5,20),(43,300)), fill=(70,70,70))
police = ImageFont.truetype('Polices/KGPrimaryItalics.ttf',36)
zone_ecran.textrotated((8,35), 'Département GEII', 270, font=police, fill=(255,203,0))
```

La partie centrale n'est nécessaire que parce qu'il y a une transparence à gérer dans le logo et que celle-ci n'est pas gérée par la librairie de l'écran.

La partie finale où l'on affiche le texte est facile à comprendre :

- On crée un rectangle gris (pour que le texte soit plus lisible)
- On charge une police dans l'objet 'police'
- On affiche un texte avec la méthode 'textrotated()' où les paramètres sont :
  - (x,y) : la position du texte
  - Le texte souhaité
  - L'angle de rotation du texte
  - La police utilisée
  - La couleur de remplissage du texte

Contrôler que votre affichage est correct avant de passer à la suite.

QUELQUES NOTIONS LIÉES AU WEB :

Le titre de ce TP est '**utilisation d'APIs**', mais qu'est-ce qu'une API ?
En fait, vous en utilisez tous les jours car votre téléphone est un très grand consommateur d'APIs.
L'acronyme **API** signifie '**Application Programming Interface**' soit en français '**interface de programmation d'application**'

Selon **Wikipedia**, voici la définition d'une API :

*« En informatique, une **interface de programmation d'application**[1] ou **interface de programmation applicative** (souvent désignée par le terme **API** pour application programming interface) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.*

Dans l'industrie contemporaine du logiciel, les applications informatiques se servent de nombreuses interfaces de programmation, la programmation se fait en réutilisant des briques de fonctionnalités fournies par des logiciels tiers. Cette construction par assemblage nécessite pour le programmeur de connaître la manière d'interagir avec les autres logiciels, qui dépend de leur interface de programmation. Le programmeur n'a pas besoin de connaître les détails de la logique interne du logiciel tiers, et celle-ci n'est généralement pas documentée par le fournisseur. »

Donnons tout de suite un exemple pour comprendre :
Vous êtes passionné de photographie et vous avez créé un site web où vous proposez en visualisation vos plus belles œuvres.

- En plus d'une petite légende, vous aimeriez associer à chaque photo de votre site une carte dynamique avec un marqueur pour indiquer le lieu où elle a été prise. La solution la plus simple consiste à intégrer un service web qui fournit déjà des cartes comme Google Maps, Bing Maps ou OpenstreetMap. Pour accéder aux fonctionnalités et aux données d'un tel service, vous allez pouvoir le faire via une API. Bien sûr, il faut que cette dernière soit correctement documentée pour savoir comment l'utiliser. Tous les services web officiels proposant une API, offrent une documentation adéquate. Pour notre exemple, voici les liens :
  - https://developers.google.com/maps/documentation/?hl=fr
  - https://www.microsoft.com/en-us/maps/choose-your-bing-maps-api
  - https://wiki.openstreetmap.org/wiki/API_v0.6
- Et puis, parce que vraiment votre travail est reconnu, vous aimeriez arrondir un peu vos fins de mois en vendant quelques photos. Pour cela, vous voulez faciliter et sécuriser les choses en choisissant Paypal. Là aussi, il vous faudra utiliser leur API pour pouvoir intégrer correctement ce mode de paiement dans votre propre site. Et là aussi, la documentation est évidemment fournie :
  - https://developer.paypal.com/docs/api/overview/

Il existe un nombre incroyable d'API disponibles : Facebook, Twitter, Instagram, Whatsapp, Alexa, Gmail, Github, Youtube, Radio France, Météo France, … Même l'état français fournit à l'heure actuelle un portail vers un nombre conséquent d'APIs (https://api.gouv.fr).
C'est par exemple grâce aux APIs des différentes compagnies aériennes qu'un site comme KAYAK peut vous fournir une comparaison des différents vols pour une destination.
Certaines APIs sont gratuites, partiellement payantes ou totalement payantes. Elles nécessitent souvent l'authentification de l'utilisateur du service via une clef numérique.
Par exemple, Google Maps a fourni une API gratuite pendant des années et depuis juillet 2018, elle est devenue partiellement payante (limite de gratuité jusqu'à 200$). Si vous créez une application qui 'interroge' Google Maps pour afficher des cartes statiques, vous ne pourrez le faire gratuitement que 100.000 fois par mois (https://cloud.google.com/maps-platform/pricing/?hl=fr). S'il n'y a que vous qui l'utilisez, cela ira, mais si vous avez 30.000 utilisateurs (ce qui n'est pas aberrant pour un site web ou une appli mobile), ils ne pourront le faire qu'environ 3 fois par jour chacun !!

Avec un Raspberry Pi, si l'on veut créer un code Python qui utilise une API, il y a plusieurs cas qui peuvent se produire :

- Une librairie Python est fournie par le service qui a développé l'API (exemple : Google Maps) ou par un tiers qui l'a créée (exemple autour de l'API des mots officiels au Scrabble : https://github.com/fogleman/TWL06). Voici notamment une liste non exhaustive de librairies API Python : https://github.com/realpython/list-of-python-api-wrappers
- Aucune librairie Python n'est fournie, mais des fichiers d'échanges de données sont fournis au format XML, JSON, YAML, … (exemple : http://fr.openfoodfacts.org/ qui classe des données sur les produits alimentaires d'après leurs codes barre)

Nous allons étudier ces deux cas de figures durant cette séance et celle à venir.

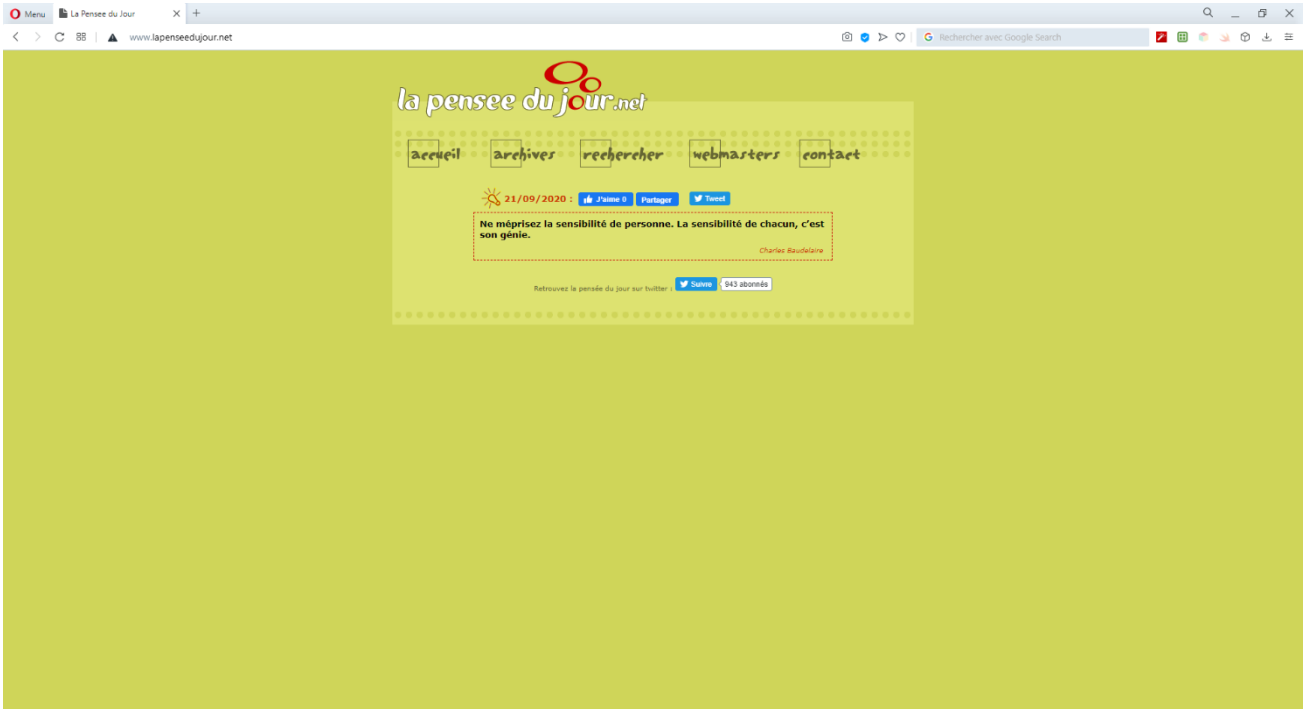
Mais il existe enfin une dernière situation où aucune librairie Python n'est fournie et où aucun fichier et aucune documentation n'est disponible non plus et malgré tout, on veut récupérer des informations issues du web. C'est ce dernier cas que nous allons étudier tout de suite.

RECUPÉRATION DE DONNÉES ISSUE DU WEB SANS API

EXERCICE 1 : 'Ex01.py'

Quand aucun fichier ou aucune librairie ne nous permet d'accéder à des données du web, il y a la solution du '**scraping**', c'est-à-dire l'extraction de données depuis le code source **HTML** d'une page web.

On va prendre un exemple concret dès maintenant pour mieux comprendre ce type de situation.
Ouvrez un navigateur et allez à l'adresse suivante : https://www.lapenseedujour.net



Il s'agit d'un site qui fournit chaque jour une pensée, une citation différente. On va chercher ici à récupérer le texte de cette citation pour l'afficher sur notre écran LCD.
Bien évidemment, le webmaster de ce site n'a pas pris la peine de développer une API ou de mettre en ligne un fichier actualisé chaque jour avec la citation correspondante.
Peut-être même exploite-t-il lui aussi le contenu d'un autre site pour afficher cette citation…

Il va donc falloir récupérer cette information par nous-même… avec l'aide du langage Python tout de même !!
Cette opération s'appelle '**web scraping**' ou '**web harvesting**'. Voici la définition du journal du net à ce sujet : https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203521-scraping-definition-traduction/

Effectivement, on va, en ce qui nous concerne, créer un script Python qui va extraire à partir du code html de cette page web, uniquement la citation du jour.

Et ce script permettra toujours cela, même demain, après-demain ou dans un an pour peu que ce site soit toujours actif et qu'il ne bloque pas l'accès à son contenu. En effet, bon nombre de sites empêchent maintenant l'accès automatisé à leur contenu pour éviter par exemple la veille concurrentielle (e-commerce) ou l'extraction de base de données ou encore l'extraction de 'data sets' pour le Machine Learning.

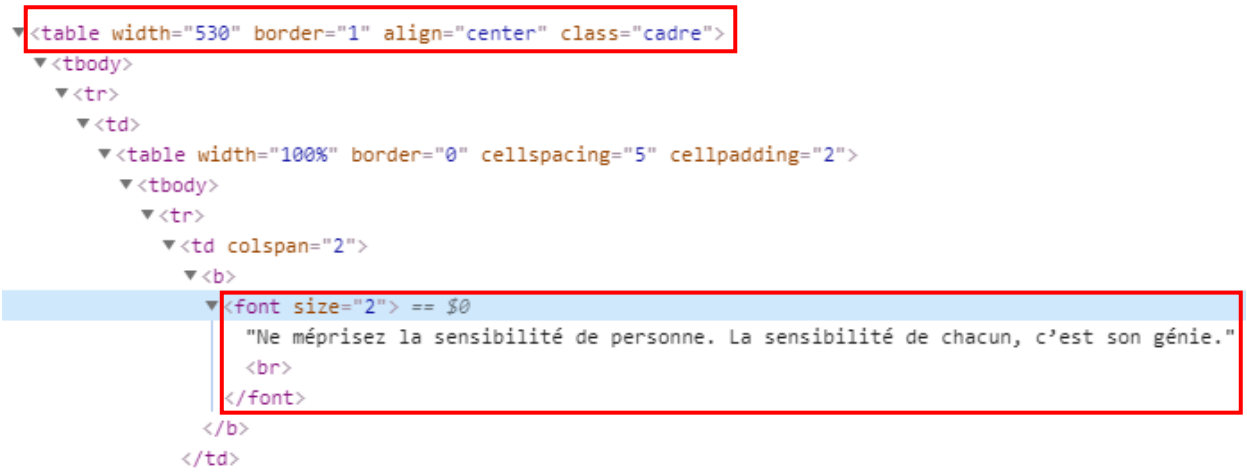
Toutefois, pour simplement extraire le texte de cette pensée de façon automatique avec Python, il serait difficile de vous laisser trouver une solution par vous-même même avec l'aide de quelques informations.

Par conséquent, je vais ici vous indiquer toute la démarche et le code pour aboutir à ce résultat. Vous pourrez de votre côté vous en inspirer pour reproduire ce résultat sur un autre site.

La première étape consiste à explorer tout le code '**HTML**' de la page concernée. N'importe quel navigateur propose cette option d'afficher le code source de la page. Toutefois, un outil bien plus pratique est présent sur la plupart des navigateurs : **l'inspecteur d'éléments**.
Sélectionnez le texte de la citation et faîtes un clic droit dessus et sélectionnez un élément du menu qui ressemble à :

Voici alors le type d'informations qui s'affiche dans le navigateur :

Vous retrouvez des balises '**head**', '**body**'… que vous avez déjà utilisées lors du TP n°4 ainsi que beaucoup d'autres puisque cette page est bien plus complexe que celles que nous avons déjà créées précédemment.
Il faut repérer précisément où se situe l'information qui vous intéresse :



Pour extraire le texte, on va utiliser les balises pour le retrouver dans le flot du code.
Toutefois, si l'on s'intéresse à des balises de type 'tr' (ligne de tableau), 'td' (cellule de tableau), elles sont trop nombreuses. Il faut rechercher des balises moins communes et si possible ayant des attributs précis.

Sans réellement tout comprendre, on voit notamment que notre pensée est dans une balise **<table>** (un tableau en html) avec une classe (style CSS du tableau) appelée **'cadre'**. Cette balise est unique dans le code html. Elle est donc facilement identifiable.
Toutefois, si l'on extrait tout ce qu'il y a entre les balises **<table>** et **</table>**, il nous reste encore beaucoup trop d'informations ! On peut donc affiner l'extraction en précisant que notre texte est ensuite entre les balises **** et ****.
Mais comment faire tout cela en Python ?

Pour nous simplifier la tâche, il existe une librairie python au nom humoristique qui va nous permettre d'extraire du contenu dans un code HTML : « **BeautifulSoup** » !!

La première chose à faire est donc de l'installer via un terminal (elle est normalement installée par défaut mais il est toujours utile de le vérifier) :
$ sudo pip3 install BeautifulSoup4

Ouvrez ensuite dans **Thonny** le fichier '**Ex01.py**'. Le contenu est le suivant :

```
1  from bs4 import BeautifulSoup
2  import requests
3
4  URL_site = 'https://www.lapenseedujour.net'
5
6  # Récupération de la page
7  contenu_brut = requests.get(URL_site)
8  # Trie le contenu html pour le stocker
9  code_html = BeautifulSoup(contenu_brut.text, 'html.parser')
10 #print(code_html)
11
12 information_recherchee = code_html.find('table', attrs={"class":u"cadre"}).find('font')
13
14 # extraction du texte
15 pensee = information_recherchee.text
16
17 print(pensee)
```

Testez ce code pour vérifier que dans la zone 'console' de Thonny, vous affichez bien la pensée du jour.

J'ai mis quelques commentaires dans le code, mais je vais maintenant vous fournir quelques explications supplémentaires.

- La ligne 7 utilise le module '**requests**' qui permet d'aspirer le contenu d'une page web dont on a spécifié l'URL. L'objet '**contenu_brut**' contient déjà le code HTML
- La ligne 9 nous permet de trier et de mettre en forme le code HTML récupéré.
- La ligne 13 permet de rechercher la balise '**<table>**' dont l'attribut est '**class=''cadre'''**' puis à l'intérieur la première balise '**<font>**' rencontrée.
- Enfin la ligne 16 extrait du code contenant la balise '**<font>**' uniquement le texte, soit notre citation.

Remarque : c'est bien évidemment la méthode '**find()**' qui est particulièrement importante. C'est une méthode très puissante qui permet une recherche très précise ou très complète.
On pourrait par exemple récupérer toutes les balises cellules de la page par la commande :
code_html.find_all('td')

Python nous renverrait alors une liste (i.e. l'équivalent d'un tableau) contenant l'ensemble des cellules séparées par des virgules.
Pour accéder au texte du 5ème élément de cette liste, on aurait juste à utiliser la méthode comme ceci :
code_html.find_all('td')[4].get_text()

C'est maintenant à vous de travailler.
Vous allez compléter le code précédent pour afficher sur l'écran LCD de la carte d'extension notre citation du jour.
Vous devez respecter la mise en forme imposée sur la page suivante.

Voici ce que vous devez obtenir sur votre écran :

Il y a de grandes chances que le texte complet de la citation ne tienne pas sur la largeur de l'écran. Il faudra donc l'afficher sur plusieurs lignes.
Heureusement python dispose d'un module '**textwrap**' qui vous permettra de formatter votre texte avec un nombre de caractères par ligne défini.
Pour l'utiliser, je vous conseille de lire l'exemple de cette page : https://www.dotnetperls.com/textwrap-python

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

RECUPÉRATION DE DONNÉES ISSUE DU WEB VIA UN FICHIER CSV

EXERCICE 2 : 'Ex02.py'

Dans ce second exemple, on va récupérer un fichier fourni par un site web et en extraire les informations qui nous intéressent.

Comme cela a été évoqué, certains sites proposent en ligne des informations statiques ou dynamiques via un fichier d'échange de données. Les formats les plus couramment utilisés pour ces fichiers sont : RSS, XML, CSV, JSON, YAML, …
On va ici récupérer et traiter un fichier **CSV**.
Ce fichier est fourni par **La Poste** : il s'agit de la liste mise à jour de toutes les communes françaises avec leur code postal associé.

On va donc voir maintenant comment récupérer ce fichier et en extraire le code postal d'une commune, le nom de cette commune ayant été demandé à l'utilisateur au démarrage du programme.

Ouvrez le fichier '**Ex02.py**' dans **Thonny**. Son contenu est le suivant :

```
1  # Importation des librairies
2  import requests
3  import csv
4
5  # Récupération du fichier officiel des codes postaux s'il n'existe pas
6  try :
7      with open('codes_postaux.csv') :
8          pass
9  except IOError :
10     url = 'https://datanova.legroupe.laposte.fr/explore/dataset/laposte_hexasmal/download/
11     mon_fichier = requests.get(url)
12     with open('codes_postaux.csv', 'wb') as donnees :
13         donnees.write(mon_fichier.content)
14
15 # Demande à l'utilisateur le nom de la commune
16 commune = ???????
17 # Mise en majuscules
18 commune = commune.upper()
19 # Affichage du nom de la commune dans la console
20 print(commune)
21
22 # Extraction du code postal dans le fichier CSV
23 fichier_csv = open("codes_postaux.csv", "r")
24 try :
25     lecteur_csv = csv.reader(fichier_csv, delimiter=";")
26     for ligne in lecteur_csv :
27         ?????????????
28         # Affichage du code postal dans la console
29 finally :
30     fichier_csv.close()
```

Vous devez compléter légèrement ce code avant de pouvoir le tester.

Quelques indications pour cela :

- **Lignes 2-3 :** la librairie **'requests'** est la même que pour l'exercice n°1, elle permet de faire des requêtes sur le web, de récupérer une page web ou un fichier d'échange de données dans notre cas.
- **Lignes 6-13 :** On essaye d'ouvrir le fichier **'codes_postaux.csv'** s'il existe dans le dossier du programme, sinon on lève une exception. Et dans cette exception, on va récupérer le fichier en ligne et on le sauvegarde dans le dossier du programme. **ATTENTION**, soyez patient lors de l'exécution du code car la récupération du fichier (presque 3Mo) est un peu longue !!
- **Ligne 16 :** vous devez demander le nom de la commune à l'utilisateur via la console
- **Ligne 18 :** comme dans le fichier **'codes_postaux.csv'**, les noms de communes sont en majuscules, on change la casse du texte de l'utilisateur pour le mettre qu'en majuscules.
- **Ligne 23 :** on ouvre le fichier **'codes_postaux.csv'** en lecture seule

- **Lignes 26-28 :** Avec une boucle for, on parcourt chaque ligne du fichier **'codes_postaux.csv'**. Ainsi l'objet **'ligne'** contiendra une liste dont chaque élément correspond au contenu de chaque colonne de la ligne en cours de lecture. Regardez donc la structure du fichier **'codes_postaux.csv'** (en l'ouvrant avec Libre Office) pour visualiser dans quelles colonnes récupérer le nom et le code postal de la commune et compléter le code.
- **Ligne 30 :** on ferme le fichier **'codes_postaux.csv'** lorsque l'on a trouvé le code postal.

Une fois que vous avez complété ce code, vous pouvez déjà tester son bon fonctionnement avant de passer à la suite.

Maintenant, on va ajouter les lignes de code nécessaires qui permettront d'afficher le nom de la commune et le code postal attendu sur l'écran graphique LCD de la carte d'extension.

Voici à quoi doit ressembler votre affichage :

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

RECUPÉRATION DE DONNÉES ISSUE DU WEB VIA UN FLUX DE DONNEES RSS

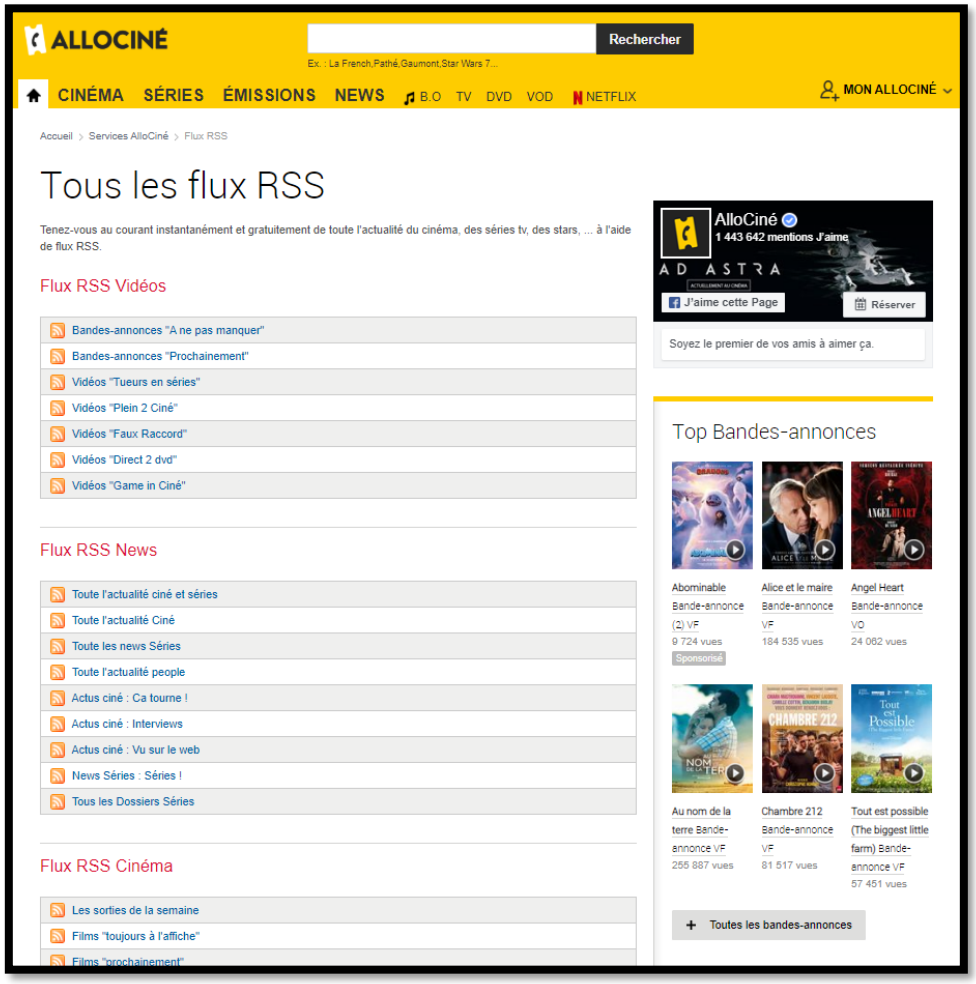
EXERCICE 3 : 'Ex03.py'

On va terminer cette séance avec un dernier exemple.
Pour ne pas avoir à consulter régulièrement un site pour savoir s'il y a du nouveau ou pour être au courant de la dernière promo du moment, les concepteurs de site disposent principalement de trois outils :

- La newsletter
- Le flux RSS
- Les réseaux sociaux

On va ici s'intéresser à **RSS** ('Really Simple Syndication' : publication vraiment simple) qui est apparue pour la première fois en 1999. C'était une des premières solutions permettant de suivre automatiquement un fil d'actualités en ligne autour de différents sujets par le biais d'un agrégateur de flux.
Ce format est encore très présent sur le web. Vous pouvez facilement identifier les sites qui le proposent par le biais de cette icône :

On va dans cet exercice exploiter un flux RSS d'actualité cinématographique.
Ouvrez un navigateur et allez consulter la page : http://www.allocine.fr/xml/rss/



Vous pouvez constater qu'une vingtaine de flux d'actualité RSS sont disponibles. Si vous cliquez sur l'un d'entre eux, vous aboutirez à une nouvelle page web mais il s'agit en fait du flux RSS mis au format HTML par un agrégateur de flux généralement intégré à votre navigateur internet, car le flux RSS est plutôt un fichier au format XML.

Nous allons donc écrire un script en python qui nous permet de récupérer les titres des films qui sortent cette semaine au cinéma. Le flux concerné est à l'adresse : http://rss.allocine.fr/ac/cine/cettesemaine

Comme précédemment, je vais vous fournir le code, mais au préalable, on a besoin d'une librairie python qui va nous permettre d'extraire les données incluses dans un flux RSS.

Celle que l'on va utiliser est la plus usitée : '**feedparser**'

Ouvrez donc un terminal et installez cette librairie via la commande suivante :
$ sudo pip3 install feedparser

Ouvrez ensuite le fichier '**Ex03.py**' dans Thonny :

```
1  # Importation des librairies
2  # sudo pip3 install feedparser
3  import feedparser
4
5  # Récupération du flux allocine des sorties de la semaine
6  url = 'http://rss.allocine.fr/ac/cine/cettesemaine'
7  flux = feedparser.parse(url)
8  #print(flux)
9
10 # Extraction des informations
11 entrees_flux = flux.entries
12
13 for entree in entrees_flux :
14     print (entree.title)
```

Vous pouvez tester ce code immédiatement pour vérifier qu'il correspond bien à ce que vous avez visualisé dans votre navigateur.

Décrivons rapidement ce qu'il contient :

- **Ligne 3 :** on importe la librairie qui va permettre d'extraire les différentes informations contenues dans le flux RSS
- **Lignes 6-7 :** on récupère le flux RSS des sorties ciné de la semaine
- **Ligne 8 :** permet d'afficher tout le flux (pour information)
- **Ligne 11 :** on récupère chaque section (donc ici chaque film) dans le flux
- **Lignes 13-14 :** on extrait dans chaque section récupérée le titre de la section qui va correspondre ici au titre du film

On pourrait se contenter de ce résultat et afficher les différents titres des films et documentaires sur l'écran LCD, mais on va aller un peu plus loin.
On va chercher à faire défiler les films en boucle en ayant à chaque écran le titre du film ainsi que son affiche.

On a déjà récupéré les titres, il nous faut maintenant télécharger les affiches. Voyons rapidement les étapes nécessaires avant de compléter l'écriture du code python :

1. Sur la page web affichant le flux RSS, vous pouvez constater l'existence pour chaque film d'un lien pointant vers un fichier image jpg :

2. Si vous cliquez dessus, vous aboutirez à l'affiche du film. Il s'agit d'une base de données en ligne permettant aux différents média (ciné, TV, journaux, …) d'accéder notamment aux affiches des films pour les utiliser sur leur support. Tout comme on l'a fait pour les titres, il va donc falloir récupérer l'ensemble de ces URLs (liens).
3. Ensuite pour chaque lien, il va falloir télécharger le fichier jpeg associé et le stocker dans un dossier (pour nous le dossier 'affiches').
4. Comme chaque image est de résolution différente et surtout d'une résolution supérieure à notre écran, il va falloir les redimensionner avant de pouvoir envisager de les afficher.

Dans votre code, vous allez commencer par ajouter une librairie : **import shutil**
Ce module python fournit des méthodes de haut niveau pour la manipulation des fichiers et des dossiers (renommer, copier, déplacer, …)

Ensuite, après avoir récupéré le flux RSS, vous allez pouvoir ajouter le code suivant :

```
# Extraction des informations
entrees_flux = flux.entries
titres_films = []
affiches_films = []
index = 0
for entree in entrees_flux :
    #print (entree.title)
    titres_films.append(entree.title)
    #print (entree.links[1]['href'])
    affiches_films.append(entree.links[1]['href'])
    affiche_fichier = requests.get(affiches_films[index], stream=True)
    fichier = open("affiches/" + str(index) + ".jpg", 'wb')
    affiche_fichier.raw.decode_content = True
    shutil.copyfileobj(affiche_fichier.raw, fichier)
    fichier.close()
    affiche = Image.open('affiches/' + str(index) + '.jpg')
    affiche = affiche.rotate(-90, expand = True)
    affiche.thumbnail((190,190))
    affiche.save('affiches/' + str(index) + '.jpg')
    index = index +1
```

Vous pouvez tester votre code et vérifier son bon fonctionnement en allant consulter le contenu du dossier **'affiches'** dans votre explorateur (ne modifiez pas du tout ces fichiers !!).

A la sortie de cette boucle **'for'**, vous disposez donc :

- D'une liste (i.e. un tableau en python) '**titres_films**' qui contient tous les titres des différents films qui vont sortir cette semaine.
- D'un dossier '**affiches**' qui contient l'ensemble des affiches de ces films numérotées en commençant par '**0.jpg**'.
- Ainsi le titre du 3ème film de la liste est contenu dans '**titres_films[2]**' et son affiche porte le nom de fichier '**2.jpg**' dans le dossier '**affiches**'.
- Pour connaître le nombre de films de la liste, il suffit de récupérer le retour de la méthode '**len(titres_films)**'.

Remarque : la plupart des images sont au format 'jpg' mais parfois une ou deux au format 'png' se glissent dans les affiches de la semaine. PIL gère parfaitement leur affichage mais il faut en tenir compte lors du chargement de l'image pour l'afficher sur le LCD.

Vous allez maintenant pouvoir terminer ce code par vous-même en vous occupant de la partie affichage sur le LCD.

Comme le téléchargement des affiches et la mise en forme des affiches prend un peu de temps, vous allez commencer par afficher un écran d'attente :

Image de fond :
'Allocine.jpg'
Dans le dossier 'images'

Police :
'DK_Pimpernel.otf'
Hauteur : 36
Couleur : (0,0,0)

Ensuite, il faut créer une boucle infinie (si possible avec un '**try… except KeyboardInterrupt**') dans laquelle vous affichez pour chaque film un écran de ce type pendant deux secondes :

Image du dossier 'affiches'

Police :
'DK_Pimpernel.otf'
Hauteur : 36
Couleur : (255,255,255)

Faîtes vérifier le fonctionnement de votre code par un enseignant.

Le format **XML** est également un format de fichier d'échange sur le web très utilisé, mais comme le flux **RSS** se base sur du **XML**, on ne va pas faire un nouvel exemple avec du **XML** dans ce TP.
Il reste tout de même un format de fichier d'échange qui est de plus en plus répandu sur le net, il s'agit du format **JSON**, mais on en reparlera à la prochaine séance de TP.