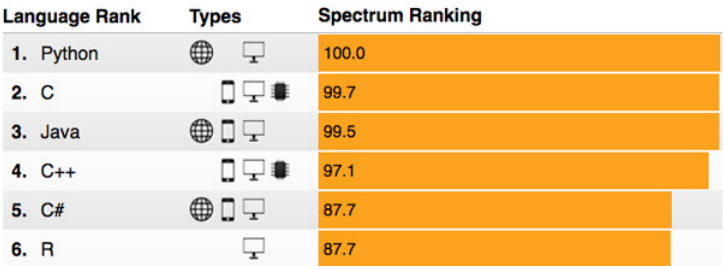
3 Le couple Arduino - Python

3.1 Pourquoi Python ?

3.1.1 Avantages pour les scientifiques

- Facile à installer, libre et multi-plateformes (Linux, Windows, macOS)
- Prise en main très rapide (quelques jours)
- Alternative fiable à des logiciels spécialisés (matlab, excel, libreOffice...)
- Spécialisé dans le calcul scientifique, la représentation des données sous forme de graphiques et la simulation
- Utilisation simple de la liaison série pour le transfert de données avec Arduino
- Python est un des langages les plus populaires d'après L'Institute of Electrical and Electronics Engineers (IEEE) qui est la plus grande association mondiale de professionnels techniques (IEEE Spectrum). Un article intéressant à lire à ce sujet : IEEE : Python devient le meilleur langage en 2017 en dépassant C et Java



3.1.2 Avantages pour les élèves

- Un document ressources de l'éducation nationale mentionnant clairement Python, vient de paraître pour la rentrée 2017 sur le thème algorithmique et programmation
- Python est très majoritairement utilisé dans l'enseignement de spécialité ISN en terminales S.
- Python est un enseignement obligatoire en C.P.G.E depuis la rentrée 2013
- L'informatique avec Python est une épreuve obligatoire des concours aux grandes écoles que ce soit sous forme d'une épreuve de simulation pour la physique - chimie (Concours communs polytechniques et e3a) ou d'une épreuve d'informatique pour tous plus théorique (Centrale-Supélec et Mines-Ponts)

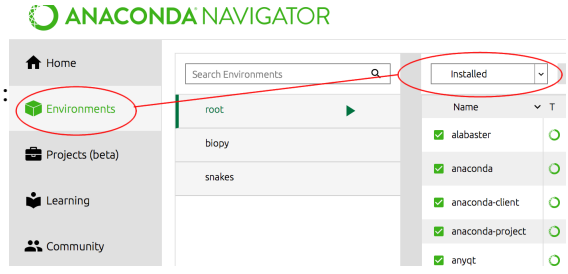
3.2 Installation de Python

Préparer la formation : Téléchargement d'Anaconda 3.xx puis Installation sur le site officiel en anglais. Il semble que les versions récentes d'Anaconda **ne contiennent pas forcement** le package **pyserial** dont nous allons avoir besoin pour communiquer avec Arduino (À tester lorsque vous aborderez l'exemple : *Lecture des données* 3.4.2).

3.2.1 Installation du package pyserial

Si lors du test vous obtenez une erreur avec le package *pyserial*, vous pouvez l'installer de différentes manières :
depuis l'interface d'Anaconda Navigator (recommandé)

- Cliquer dans la fenêtre d'accueil sur le menu : **Environments**.
- Un menu déroulant en haut de la fenêtre indique par défaut : **installed**
- Choisir **Not installed**
- Sélectionner dans la liste le module **pysérial**.
- Valider pour l'installation.



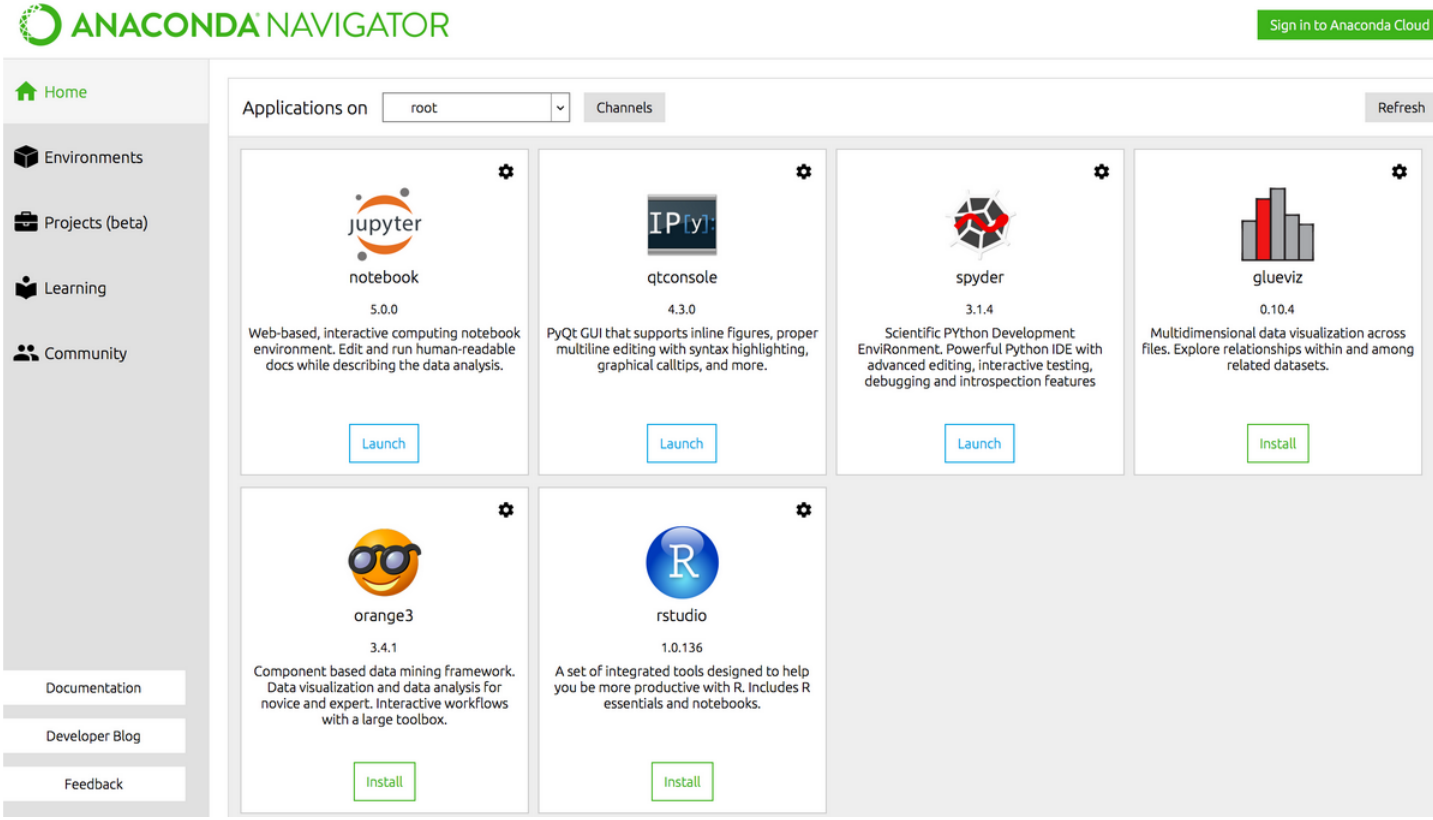
depuis Conda ou PyPI :

- Linux / macOS / Windows : PySerial

3.3 Utilisation du Jupyter Notebook

Maintenant que Python est installé sur votre ordinateur, il nous faut un environnement de programmation. Pour faire très simple, un éditeur de texte, permettant d'écrire et d'interpréter du code Python (et bien plus...). Pour cela nous allons utiliser le Jupyter Notebook. Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation. Jupyter permet de réaliser des notebooks, c'est-à-dire des feuilles de programmes contenant à la fois du texte (en markdown), du code Python et pour les connaisseurs vous pourrez même insérer du code LaTeX pour rédiger de belles équations. Ces notebooks sont très utilisés en science pour explorer, analyser et présenter des données. Exemple de notebook pour le test d'un capteur infrarouge.

Pas de panique le Jupyter Notebook est présent dans la distribution Anaconda que vous venez d'installer. Elle propose un bureau graphique appelé Anaconda Navigator. Il ne reste plus qu'à lancer Jupyter Notebook avec le bouton **Launch**.



Pour bien démarrer voici un petit guide Jupyter Notebook pour en savoir un peu plus sur le Jupyter Notebook.

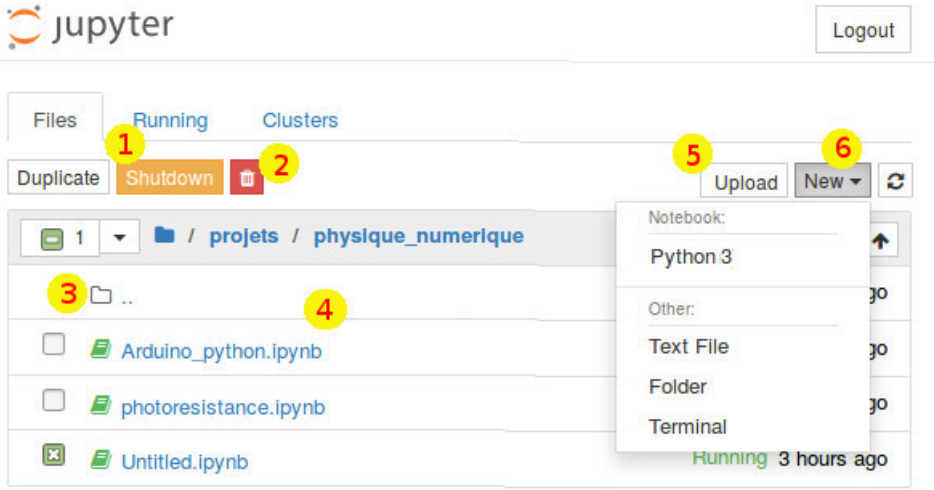
Remarques :

- Quand vous exécutez le programme *Jupyter Notebook*, une fenêtre avec un fond noir s'affiche à l'écran, elle permet d'observer les commandes qui lancent les services (noyau Python, navigateur web, ...) du *Jupyter Notebook*. Surtout ne pas la fermer.

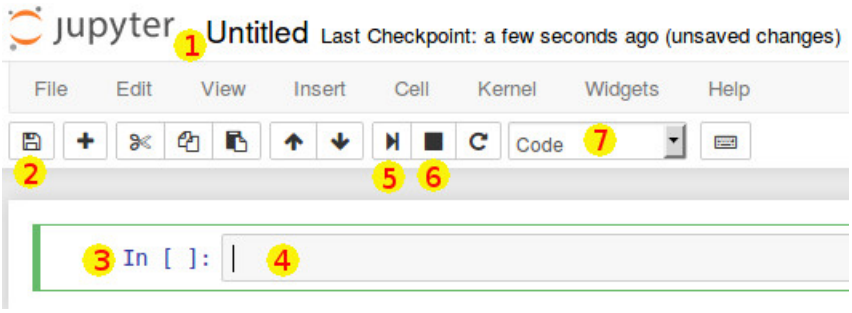
- Si tout se passe bien Jupyter lance un navigateur web servant d'interface graphique pour la programmation Python. Il se peut que la première fois le lancement soit relativement long. Un jour une élève est venue me voir pensant que l'installation n'avait pas marché. Après un rapide état des lieux de sa machine, j'ai constaté qu'il fallait plusieurs minutes à Windows pour lancer son antivirus (Avast) ce qui décalait d'autant le lancement d'Anaconda Navigator. Donc dans certains cas patience...

Vous pouvez également travailler à partir d'une version en ligne : try.jupyter. Attention cette version en ligne ne permet pas d'utiliser le package `pyserial`. Mais elle reste tout de même très performante pour travailler avec des élèves.

3.3.1 Description sommaire de l'interface web du notebook



1. Fermer un notebook
2. Effacer un notebook
3. Dossier parent
4. Liste des notebooks, cocher pour sélectionner un notebook
5. Charger un notebook
6. Créer un nouveau notebook avec Python 2 ou 3 suivant les versions



1. Clic gauche pour changer le titre (Untitled) du notebook.
2. Sauvegarder le notebook
3. Cellule du notebook
4. Zone de code python
5. Exécuter le code (ou *shift* + *enter*)
6. Stopper l'exécution du code
7. Sélection du type de contenu dans la cellule en cours.

3.4 Communication Arduino - Python via le port série

3.4.1 Dans quel but ?

La carte Arduino permet de faire l'acquisition d'un signal analogique par l'intermédiaire d'un capteur et de le convertir en signal numérique grâce à son CAN (10 bits). Il est ensuite possible de transférer ces données par le port série vers l'ordinateur pour réaliser un traitement numérique avec Python.

Capteur	**Echantillonage et conversion**	**Traitement numérique du signal**
Photorésistance	Carte d'acquisition Arduino	Python

Donc très schématiquement on se sert de l'interface de programmation d'Arduino pour écrire un petit programme qui explique à la carte comment faire l'acquisition (programme qui est ensuite téléverser sur la carte par le port série) puis on récupère les données via le port série pour en faire une analyse avec Python.

3.4.2 Lecture des données envoyées par la carte Arduino avec Python

Le code Arduino ci-dessous envoie une valeur entière aléatoire toutes les secondes en passant à la ligne après chaque valeur (réponse à la question 8).

Remarque : *If it is important for a sequence of values generated by random() to differ, on subsequent executions of a sketch, use randomSeed() to initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.*

Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling randomSeed() with a fixed number, before starting the random sequence.

Code Arduino à téléverser sur la carte

```
1 void setup(){
2     Serial.begin(9600);
3     randomSeed(analogRead(0));
4 }
5 void loop(){
6     Serial.println(random(1,100));
7     delay(1000);
8 }
```

Code Python : Pour notre premier exemple, nous allons créer une liaison série pour que Python puisse communiquer avec la carte Arduino :

- **Fermer le moniteur série** coté Arduino, pour pouvoir établir une liaison avec Python
- Ouvrir un nouveau Notebook.
- Changer le nom du notebook : Arduino_Python
- Recopier l'exemple ci-dessous en n'oubliant pas d'exécuter la cellule de code. Attention de bien indiquer le port sélectionné dans le menu Arduino (`Outils -> Port série`). Sous Windows : COM suivi d'un numéro (1, 2, 3, ...), sous linux : /dev/ttyACM suivi d'un numéro (0 ou 1 en général) ou /dev/ttyUSB0

```
1 import serial
2 serial_port = serial.Serial(port = "COM1", baudrate = 9600)
3 serial_port.readline()
```

j'ai effectué une copie d'écran afin que l'on puisse avoir une vue du programme dans le Jupyter Notebook. Chaque case est ce que l'on appelle **une cellule** et l'ensemble des deux cellules forme notre programme Python. Les cellules ne sont pas indépendantes les unes des autres, **elles forment un tout**, comme si **le code avait été écrit dans une seule et même cellule**. Attention lors de la copie d'écran j'ai séparé les lignes 1 et 2 de la ligne 3 afin d'introduire quelques commentaires.
Le résultat de notre programme peut être visualisé sur la sortie standard (`out[3]:` dans le notebook) avec un nombre entier aléatoire suivi de 4 caractères indiquant la fin de ligne et le retour à la ligne, ces caractères peuvent changer en fonction du système d'exploitation : `'35\r\n'`.

Il y a également deux cellules de texte pour donner quelques explications. La mise en forme d'une cellule de texte se fait à l'aide de la syntaxe Mardown

Dans cet exemple pour obtenir une nouvelle valeur, il faut relancer à chaque fois la cellule contenant l'instruction : `serial_port.readline()`. Pour afficher plus de valeurs on peut utiliser une **structure de contrôle** appelée boucle. Si on veut dix valeurs on peut écrire :

```
1 for i in range(10):
2     print(serial_port.readline())
```

Attention le fait de valider plusieurs fois une cellule pour obtenir de nouvelles valeurs n'effectue pas un reset de la carte Arduino, contrairement au fait de fermer le moniteur série (cf 2.6.1). C'est à dire que le setup n'est pas relu. Pour s'en convaincre il suffit de tester le programme suivant :

Code Arduino à téléverser

```
int i;
void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
  i = 0;
}
void loop(){
  Serial.print(i);
  Serial.print("\t");
  Serial.println(random(1,100));
  i = i + 1;
  delay(1000);
}
```

Coté Python il n'y a rien à changer. Vous pouvez éventuellement modifier le nombre d'acquisitions. Si vous exécutez plusieurs fois le code Python vous vous apercevrez que **le compteur des valeurs transmises n'est jamais remis à zéro**.

Pour relancer la fonction *setup* du code Arduino vous devez fermer la liaison série avant d'en ouvrir une nouvelle.

Code Python pour relancer la fonction *setup* sur Arduino

```
serial_port = serial.Serial( port = "COM1", baudrate =9600)
for i in range(10):
    print(serial_port.readline())
serial_port.close()
```

Si on veut pouvoir observer une bonne synchronisation, c'est à dire récupérer les premières valeurs transmises par Arduino à partir de la réinitialision du microcontroleur et ce quelque soit la vitesse d'acquisition et de transmission, on peut utiliser le code Python suivant :

```
serial_port = serial.Serial( port = "COM1", baudrate =9600)
# réinitialisation
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
# on vide le buffer
serial_port.flushInput()
# lecture des données
for i in range(10):
    print(serial_port.readline())
serial_port.close()
```

Une des broches matérielles de contrôle du flux (DTR) du circuit intégré ATmega est connecté à la ligne de réinitialisation de l'ATmega 328 via un condensateur de 100 nanofarads. Lorsque cette broche est mise au niveau BAS, la broche de réinitialisation s'abaisse suffisamment longtemps pour réinitialiser le microcontrôleur. On force la réinitialisation juste avant la lecture des données envoyées par l'Arduino.