



Document Projet DIE Informatique

Version du 21 septembre 2022

La page Moodle, avec la FAQ, de ce cours est à l'adresse

<https://foad.univ-rennes1.fr/course/view.php?id=1010963>

Table des matières

1	Introduction.....	4
2	Une application d'affichage de données de capteurs.....	4
3	Une méthode de travail agile.....	9
3.1	L'écriture du code	10
3.2	Vidéos du projet.....	11
3.3	L'organisation en binôme	11
3.4	Comment rendre son travail	11
3.5	L'évaluation du rendu	11
3.5.1	Tableau de bord	12
3.5.2	Plateforme expérimentale de notifications.....	13
3.5.3	Notation	13
	Creuser la question	13
4	Les composants du projet.....	14
4.1	La station de travail.....	14
4.2	Le serveur web	14
4.3	Le Raspberry PI.....	14
4.4	JupyterLab	15
5	Les étapes de travail (<i>sprints</i>).....	16
5.1	Sprint 1 : Analyse du travail	17
5.1.1	Tâches.....	17
5.1.2	Rendus du sprint 1.....	17
5.2	Sprint 2 (JupyterLab) : L'utilisation de JupyterLab.....	17
5.2.1	Tâches.....	18
5.2.2	Rendus du sprint 2.....	18
5.2.3	Creuser la question	19
5.3	Sprint 3 (JupyterLab) : Le Shell	19
5.3.1	Tâches.....	19
5.3.2	Rendus du sprint 3	22
5.3.3	Quelques trucs utiles.....	22
5.3.4	Autres règles importantes	23
5.3.5	Creuser la question	23
5.4	Sprint 4 (RPI): Utiliser le Raspberry PI.....	23
5.4.1	Tâches.....	23
5.4.2	Rendus du sprint 4.....	24
5.4.3	Creuser la question	24
5.5	Sprint 5 (RPI) : L'édition de fichier texte sur le RPI	25
5.5.1	Tâches.....	26
5.5.2	Rendus du sprint 5	26
5.5.3	Creuser la question	26
5.6	Sprint 6 (JupyterLab) : Le langage Python	28
5.6.1	Tâches.....	29
5.6.2	Utilisation de l'API d'accès aux données.....	30
5.6.3	Rendus du sprint 6.....	31
5.6.4	Creuser la question	31

5.7	Sprint 7 (RPI) : La création d'un fichier CSV avec les données temps réel	33
5.7.1	Tâche.....	33
5.7.2	Rendus du sprint 7	34
5.8	Sprint 8 (JupyterLab): Le langage HTML	34
5.8.1	Tâches.....	34
5.8.2	Rendus du sprint 8.....	35
5.8.3	Creuser la question.....	35
5.9	Sprint 9 (JupyterLab) : Génération d'une carte HTML.....	36
5.9.1	Tâches.....	37
5.9.2	Rendus du sprint 9.....	38
5.9.3	Creuser la question.....	38
5.10	Sprint 10 (JupyterLab): Exercice pistes cyclables + pollution.....	38
5.10.1	Rendu du sprint.....	38
6	Résumé des rendus du projet et dates limites.....	38
7	Conclusion	39
8	FAQ	39

1 Introduction

Depuis l'époque des PC connectés par modem, les systèmes informatiques se sont fortement diversifiés et s'organisent autour de l'Internet. La plupart des applications sont distribuées sur plusieurs composants comme typiquement un smartphone et des *serveurs* dans le *Cloud*.

Le **Cloud** désigne un ensemble de ressources de stockage et de calculs accessibles par Internet. Souvent il s'agit d'ordinateurs entreposés dans un *data-center* et en accès libre-service. Réf. https://fr.wikipedia.org/wiki/Cloud_computing

L'algorithme et la programmation sont des bases fondamentales de l'informatique qui s'inscrivent maintenant dans le cadre de systèmes distribués dans l'Internet où les programmes échangent des données. Un exemple typique est le navigateur web qui s'exécute sur votre machine et échange des données avec un serveur Web.

Un **serveur Web** (ou encore serveur *HTTP*) est un serveur, connecté à Internet, spécialisé pour répondre à des requêtes du protocole *HTTP* (Hypertext Transport Protocol). Les requêtes peuvent être émises par des pages web regardées dans un navigateur (p. ex. Safari, Chrome, Firefox) ou par des applications tierces (p. ex. applications mobiles). Les services proposés par ce type de serveur permettent la mise en ligne de sites Web. De nombreux serveurs Web sont déployés dans le *Cloud*.

L'objectif de ce module d'enseignement est de vous proposer une introduction à la construction d'une application distribuée. Celle-ci permet de se familiariser avec de nombreux concepts et composants d'un système informatique moderne. Cependant, il n'est pas prévu de voir en profondeur chaque notion, ce sera fait dans la suite de vos études.

La structuration du travail qui vous est proposée est très fortement inspirée des pratiques professionnelles dans le domaine de l'informatique. La suite du document reflète cette approche. La Section 2 présente l'application d'affichage de données de capteurs que vous devez mettre en œuvre au cours de ce projet. La Section 3 décrit la méthode de travail, basée sur un processus agile, qui sera utilisée. La Section 4 donne un premier aperçu des composants informatiques du projet. La Section 5 présente les 10 étapes des travaux que vous aurez à mettre en œuvre dans les semaines qui viennent. La Section 6 indique les dates limites pour les différents rendus à faire. Une conclusion termine ce document.

2 Une application d'affichage de données de capteurs

L'application que nous vous proposons de mettre en œuvre vise à afficher dans un tableau et sur une carte des données de pollution de l'air (particules fines PM) collectées par des capteurs positionnés sur des bus rennais ainsi que dans une station fixe. La

photo d'un bus équipé est en Figure 1. Pour plus de détails concernant la collecte mobile des données référez-vous au site <http://aqmo.irisa.fr/fr/accueil/>.



Figure 1 : Bus équipé de capteurs de mesure des taux de particules fines dans l'air.

Particules Fines (PM) : Les particules fines sont des particules en suspension dans l'air (acronyme PM de Particulate Matter en anglais). Leur taille varie typiquement de 0.1 à 10 micromètre de diamètre. Dans le cadre du projet on considérera les PM10 de diamètre inférieur à 10 micromètres et les PM 2.5 de diamètre inférieur à 2,5 micromètres.

Plus d'informations :

<https://www.airbreizh.asso.fr/>
https://fr.wikipedia.org/wiki/Particules_en_suspension

Pour le projet, nous utiliserons le serveur Web **prototypel1.irisa.fr** qui servira à la mise en ligne des pages web. Par ailleurs, vous ferez vos développements avec un logiciel en ligne JupyterLab¹ (hébergé sur le serveur <https://sasdje.irisa.fr/>) ainsi que sur des nano-ordinateurs Raspberry Pi (RPI, Figure 2) pour les données produites en temps réel.

¹ <https://jupyter.org/>



Figure 2 : Raspberry PI utilisés pour le projet.

Le projet consiste à écrire des programmes pour :

- 1) capturer les données produites en temps réel sur les RPI et produire une page web avec un tableau de mesures ;
- 2) produire le code d'une page web comportant une carte avec l'ensemble des points de mesures de pollution produit par un bus.

Le processus complet de l'application est illustré Figure 4. Le nano-ordinateur reçoit des données de pollution du capteur en Figure 3. Les mesures du bus sont copiées dans le JupyterLab. Les données du bus sont géo-localisées.



Figure 3 : Capteur fixe de mesure de la pollution de l'air en particules fines.

Les pages Web produites sont publiées sur un serveur (<http://prototypel1.irisa.fr/>) qui permettra un accès Internet à ces pages.

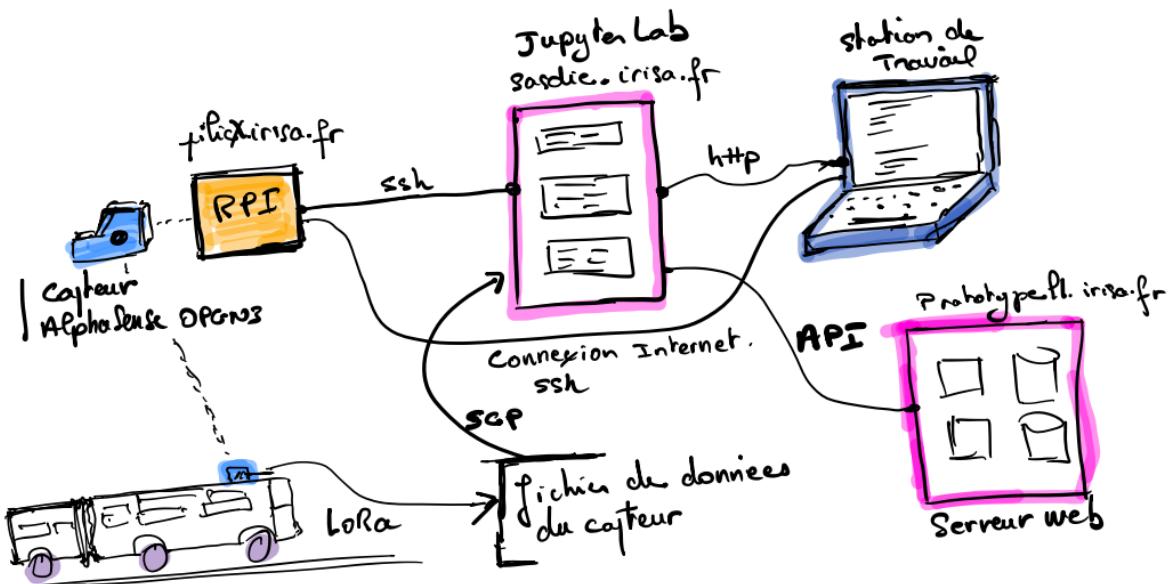


Figure 4 : Architecture du projet.

Une vidéo d'explication de la Figure 4 est ici : <https://youtu.be/UKXUVYktJSQ>

Un **programme** est un ensemble d'opérations à destination d'un ordinateur. Les programmes sont les briques de base des logiciels d'une application. Un programme est écrit dans un langage informatique.

Ref. https://fr.wikipedia.org/wiki/Programme_informatique

Ici un **code** représente le texte des instructions d'un programme. Le code peut être de différents types : source (celui qu'écrit le programmeur), binaire (celui qu'exécute la machine). Dans le contexte de ce travail, nous nous intéressons au **code source** que l'on enregistre dans des fichiers textes.

Le langage **HTML** (Hypertext Markup Language) est celui des pages Web. Ce langage est construit sur la base de balises. Par exemple, pour mettre en gros titre dans une page on écrira « <h1>Ceci est un titre en fonte large</h1> », « <h1> » est la balise ouvrante, « </h1> » la balise fermante.

Ref. https://fr.wikipedia.org/wiki/Hypertext_Markup_Language

Votre travail consistera à écrire des codes (en langage **Python version 3**) qui produiront les pages HTML nécessaires à l'affichage d'un tableau de données du capteur fixe d'une part et d'une carte d'autre part pour les données d'un bus. La connexion entre le **RPI** (pilicX.irisa.fr) et le **JupyterLab** (sasdie.irisa.fr) et le **serveur Web** (prototypel1) se fera en utilisant une API.

Une **API** (Application Programming Interface) est une interface de programmation qui offre un accès à des données ou à des méthodes de traitement. Il existe de nombreux types d'API. Dans le cadre de ce module d'enseignement, il s'agit d'un ensemble de fonctions Python permettant de communiquer avec le serveur Web.

Ceci est illustré dans la Figure 5. Le code sur le RPI utilisera les mesures des PM temps réel pour créer un fichier texte au format CSV. Ce fichier sera copié dans le JupyterLab. Dans le JupyterLab vous produirez une page Web (carte) avec les mesures mobiles du bus (qui sont géolocalisées) et une page Web (table) avec les mesures du fichier CSV. Les deux pages Web seront publiées sur le serveur « prototypel1.irisa.fr » à partir du JupyterLab.

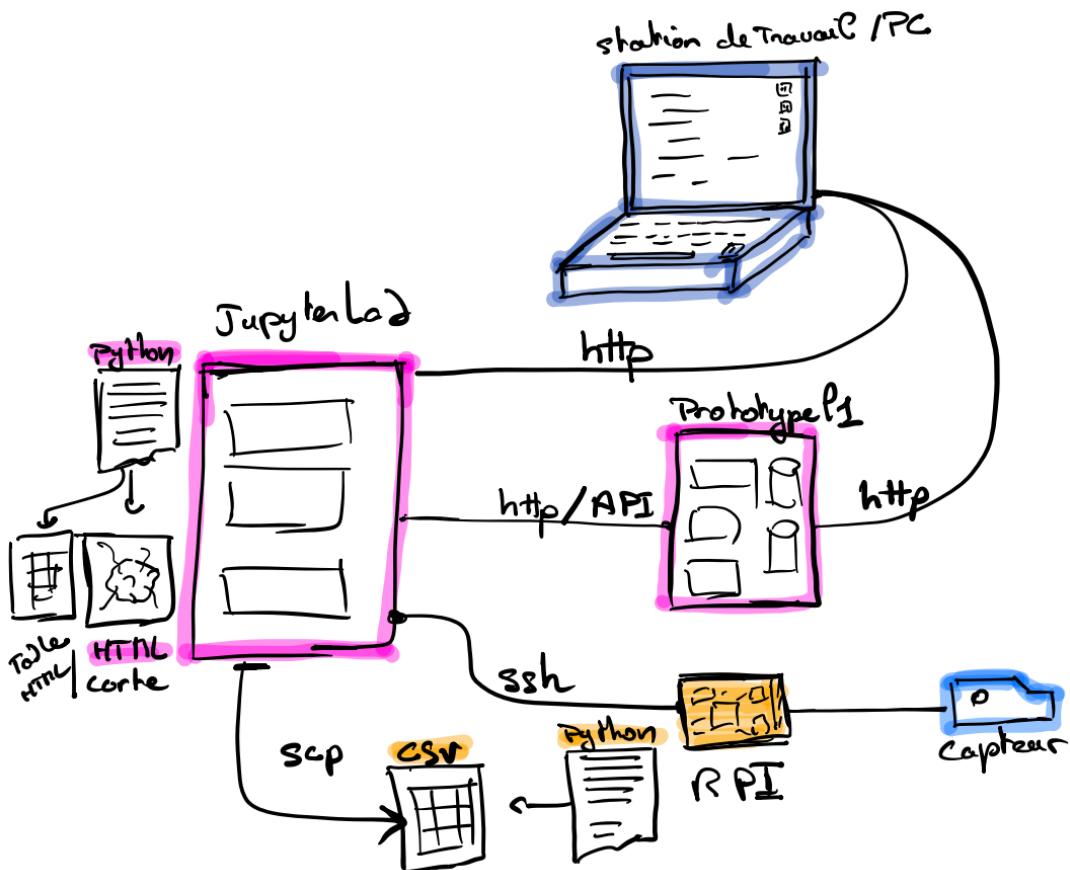


Figure 5 : Production des pages Web.

Si on résume :

1. Vous avez accès à une station de travail en salle de TP (ou à votre PC) et d'un portail JupyterLab ;
2. vous disposez d'un serveur Web qui permet le stockage et la publication de page Web (prototypel1.irisa.fr) ;
3. vous disposez d'un accès à un nano-ordinateur Raspberry PI à partir du JupyterLab. Le RPI reçoit les données en temps réel ;
4. vous devez écrire deux programmes de génération de deux pages Web, l'un pour les données d'un capteur fixe, l'autre pour les données collectées par un bus (stockées dans un fichier, `data-pm-sasdie.csv`, qui vous sera fourni).

3 Une méthode de travail agile

La méthode de travail s'inspire des techniques agiles utilisées dans l'industrie pour la mise en œuvre de projets. Cette méthode est adaptée au développement de programmes.

Les **méthodes agiles** sont des pratiques de mise en œuvre de projets fondées sur des cycles de développement itératifs. Chaque itération du cycle donne lieu à une restitution des résultats. Chaque cycle est aussi l'occasion d'adapter le travail en fonction des difficultés rencontrées. Ces méthodes sont maintenant généralisées dans la plupart des entreprises du secteur informatique.

La Figure 6 illustre un processus agile complet. Dans ce schéma, les « stories » correspondent aux fonctionnalités à mettre en œuvre. Les « *team members* » c'est vous, les « *product owners* » sont vos professeurs, les « *stakeholders* » des personnes extérieures intéressées par le projet. L'un des points très intéressants de ce processus est la boucle d'analyse rétrospective (« *sprint review* »). Nous allons en utiliser une version très simplifiée.

Ref. https://fr.wikipedia.org/wiki/M%C3%A9thode_agile

Le développement du projet se décompose en étapes (dénommées *sprint* dans la terminologie agile), chacune définie de la manière suivante :

1. les tâches à réaliser ;
2. les rendus de fichiers à effectuer.

Un **sprint** est une itération de travail d'un projet. Un sprint se compose généralement d'une réunion de concertation sur le travail à accomplir, sur la réalisation du travail et sur une démonstration des résultats obtenus.

Les sprints ont des durées variables. Vous devrez rendre vos travaux avant la date limite indiquée pour chaque sprint (voir Section 6). La régularité des rendus fait partie du contrôle de la qualité de votre travail.

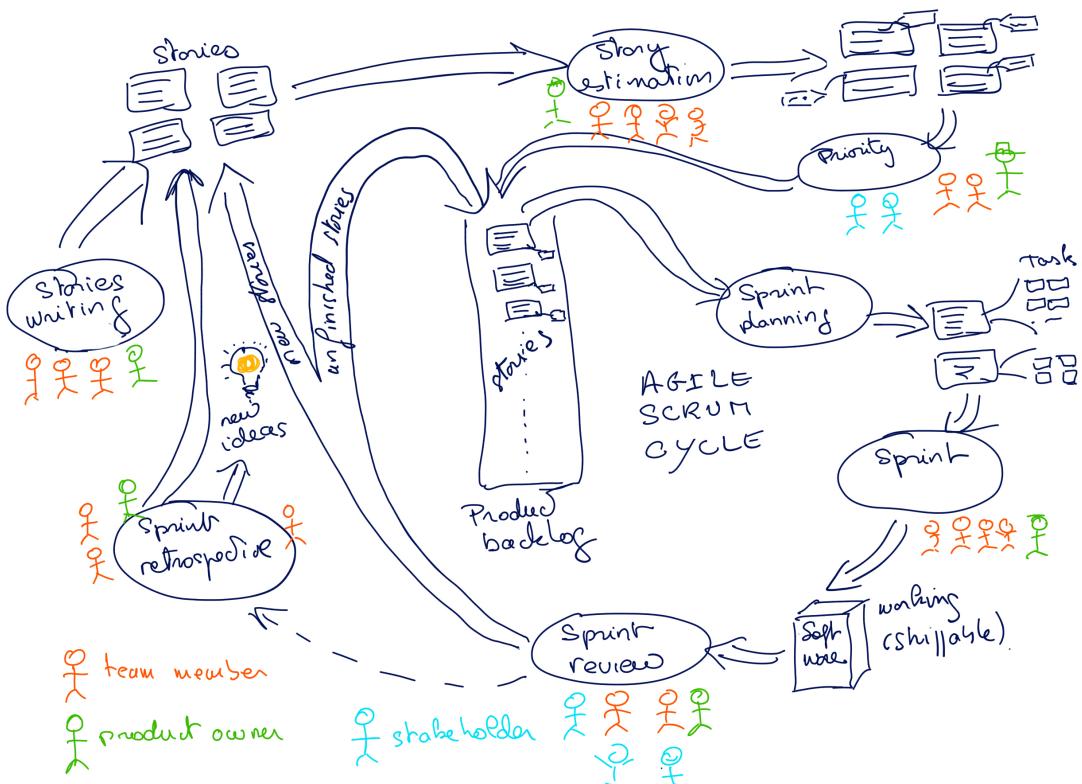


Figure 6 : Exemple de méthode agile (SCRUM).

3.1 L'écriture du code

L'écriture d'un code informatique est grandement simplifiée en suivant quelques bonnes pratiques :

1. Avant d'écrire une séquence de code, précisez les opérations à mettre en œuvre (p. ex. lister les fichiers stockés sur le serveur Web) et la manière de tester celle-ci ;
2. ne jamais écrire (ou copier-coller) un morceau de programme que l'on ne comprend pas. La technique d'essais-erreurs est peu productive ;
3. développer de manière très incrémentale en contrôlant le comportement du programme à chaque étape ;
4. prendre des notes au fur et à mesure sur ce qui est fait de manière à retrouver facilement les éléments nécessaires à la session suivante (i.e. comment se connecter, afficher telle information, etc.) ;
5. n'oubliez pas de faire des sauvegardes fréquentes de votre travail.

Quelques **bonnes pratiques** (en anglais) pour l'écriture du code sont présentées ici : « Best Practices for Scientific Computing », Greg Wilson et al. PLOS.
<http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745>

3.2 Vidéos du projet

Des vidéos expliquent la plupart des éléments techniques du projet. Ces vidéos sont accessibles à partir de Youtube. Vous pouvez les regarder pendant les séances de travaux pratiques. **Pour cela, n'oubliez pas de vous munir d'écouteurs afin de ne pas perturber les autres étudiants.** Idéalement, vous aurez regardé les vidéos avant la séance.

3.3 L'organisation en binôme

Vous effectuerez votre travail en binôme. Bien que le travail soit réalisé en binôme, **vous devrez faire un rendu individuel.**

Attention : à la fin de chaque séance, vous devrez sauvegarder les fichiers qui sont sur le Raspberry PI, car ceux-ci peuvent être réinitialisés après chaque séance.

3.4 Comment rendre son travail

À la fin de chaque sprint, il vous sera demandé de rendre vos fichiers via la page web : <http://depot-l1miee.irisa.fr/webApps/RenduSAS/>. La connexion se fait avec votre adresse mail et votre numéro d'étudiant. Quand il s'agira de rendre un code python, vous pourrez à la fin du programme ajouter directement les commandes de rendu.

[Facultatif] Par ailleurs, afin d'avoir une trace supplémentaire vous pouvez aussi envoyer par email à sasdie2019@gmail.com les fichiers. Le format de l'objet de l'email devra impérativement être :

« SASDIE – sprint *numéro* – *titre du sprint correspondant* – *sujet*».

Les fichiers transmis par email ne serviront qu'en cas de problème avec le serveur de rendu. **Utilisez l'adresse email fournie par l'université pour ces envois** (il est sinon difficile de retrouver vos messages). En cas de litige c'est ce message qui fera foi.

L'absence de rendu sera perçue comme un indice probant de décrochage.

Attention : Pendant toute la durée du module, une attention particulière sera donnée sur votre capacité à suivre les consignes et donc à la rigueur et à la régularité que vous aurez portée à la lecture des documents.

3.5 L'évaluation du rendu

L'évaluation de votre travail portera sur les éléments suivants :

1. votre présence en cours et TP sachant que les absences non justifiées conduiront à votre exclusion de ce module (i.e. 0/20), Les conditions sanitaires peuvent modifier cette contrainte ;
2. le respect des dates de rendu des travaux ;
3. la qualité du travail rendu ;

- vos réponses à l'examen final (qui aura lieu après la dernière séance de travaux pratiques).

Le rendu est à effectuer à la fin de chaque sprint. Normalement la correction est faite sous 24 heures.

Attention : lorsque vous rendez le code source d'un programme, son exécution doit se faire sans erreur. Ce problème est souvent le résultat d'une opération de copier-coller... Rendre un programme dont l'exécution ne se déroule pas correctement est sujet à pénalités.

3.5.1 Tableau de bord

La Figure 7 donne un aperçu de l'état de votre travail. Ce tableau de bord est remis à jour chaque soir. Il est disponible à l'adresse suivante :

- <http://depot-l1miee.irisa.fr/informations/>

L'état de votre travail peut être :

- 1) **validé** : c'est bon il n'y a rien de plus à faire ;
- 2) **non validé** : vous devez alors re-soumettre une version corrigée des fichiers ;
- 3) **validé avec pénalité** : vous avez rendu votre travail en retard ;
- 4) **pas de fichier** : indique que le fichier n'a pas été rendu.

Portail d'accès à vos informations

Entrez votre numéro d'étudiant:

18012310 Informations L1 Informations L2-S3 Informations L2-S4

Informations L1

Num_etudiant	18012310
Nom	Doe
Prenom	John

Informations SASDIE : John Doe 18012310

Fichiers	Date de rendu	Sprint	Penalité	Statut
typescript.txt	2018	sprint2	oui	validé
monfichier.txt	2018	sprint3	oui	non validé
amodifier.txt	2018	sprint4	oui	pas de fichier
monprogramme.lua	2018	sprint4	oui	pas de fichier
hello.lua	2018	sprint5	oui	pas de fichier
concat.lua	2018	sprint5	oui	pas de fichier
add.lua	2018	sprint5	oui	validé avec pénalité
boucle.lua	2018	sprint5	oui	pas de fichier
tableau.lua	2018	sprint5	oui	pas de fichier
liste.lua	15 nov 1961	sprint6	oui	pas de fichier
publier.lua	2018	sprint6	oui	pas de fichier
meta.lua	2018	sprint6	oui	pas de fichier
mappremierepage.html	2018	sprint7	non	pas de fichier

Figure 7 : Tableau de bord de votre travail

Vous pouvez soumettre plusieurs fois vos fichiers. Lors d'une re-soumission, le fichier courant est écrasé par le nouveau.

3.5.2 Plateforme expérimentale de notifications

Cette année nous allons mettre en place une plateforme expérimentale de notifications qui permettra à ceux qui le souhaitent :

- 1) De recevoir une notification soit par SMS soit par email si un exercice est validé ou pas ;
- 2) de recevoir des questions de quizz sur le projet afin de préparer l'examen final.

Vous recevrez par mail l'adresse d'un formulaire d'inscription à cette plateforme.

3.5.3 Notation

La notation de votre travail, sur 20, est fondée sur le barème suivant (voir plus loin la description des sprints) :

- 1) sprint1 : 1 point
- 2) sprint2 : 1 point
- 3) sprint3 : 1 points
- 4) sprint4 : 1 point
- 5) sprint5 : 1 points
- 6) sprint6 : 2 points
- 7) sprint7 : 3 points
- 8) sprint8 : 2 points
- 9) sprint9 : 3 points
- 10) sprint10 : il est noté sur 5 points, **facultatif** (pas de pénalité si non rendu).

La note finale est composée ainsi : 2/3 la note des sprints, 1/3 la note de l'examen final.

Attention : pour chaque sprint rendu en retard, une pénalité d'un point est automatiquement appliquée sur votre note finale de TP.

Creuser la question

Si vous souhaitez creuser les notions de gestion de projet abordées dans cette section, vous pouvez regarder les ressources en ligne suivantes :

- <http://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/techniques/2517/2517-gestion-de-projet-agile.pdf>

4 Les composants du projet

Le développement de l'application du projet fait appel aux éléments techniques suivants :

1. une station de travail ;
2. un serveur Web ;
3. un Raspberry PI ;
4. un serveur JupyterLab.

4.1 La station de travail

La station de travail est l'ordinateur qui sera la machine de TP ou votre PC. Elle vous permettra de vous connecter au Raspberry PI et au JupyterLab. Vous devrez disposer d'un compte ENT de Rennes 1.

4.2 Le serveur web

Le serveur Web utilisé pour ce travail est hébergé à l'Irisa (<http://www.irisa.fr/>), le laboratoire de recherche en informatique de l'Université de Rennes 1. Ce serveur est une machine qui exécute un logiciel libre appelé *Apache HTTP Server*. Ce logiciel permet de mettre en œuvre des sites Web. Le rôle du logiciel Apache est de répondre à des requêtes HTTP et de servir des pages Web (entre autres).

Ref. https://fr.wikipedia.org/wiki/Apache_HTTP_Server

4.3 Le Raspberry PI

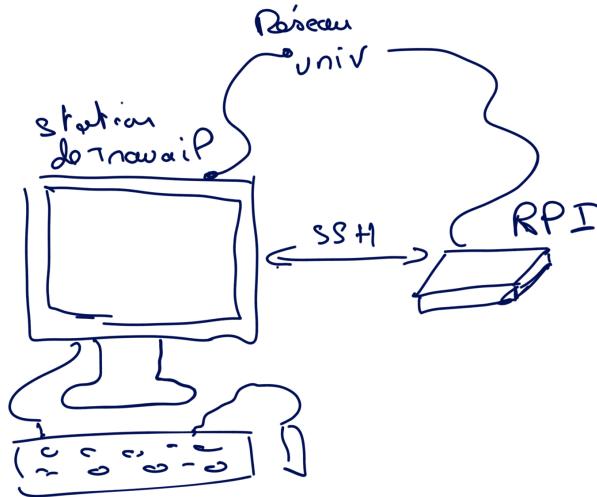
Le Raspberry PI (RPI) est un nano-ordinateur qui fonctionne avec le système d'exploitation Linux (ici la distribution est Raspbian). Nous allons utiliser un RPI sans clavier ni écran et donc à travers une connexion à distance (ici via le protocole *SSH*) à partir du JupyterLab.

Le **système d'exploitation** d'un ordinateur est l'ensemble des logiciels qui permettent de gérer les ressources d'un ordinateur (p. ex. fichier, affichage, périphériques tels que clavier ou souris ...).

Ref. https://fr.wikipedia.org/wiki/Système_d%27exploitation

Une **connexion à distance** entre ordinateurs permet à un utilisateur sur une machine de se connecter sur une autre machine. La connexion que nous allons utiliser est basée sur le protocole *SSH*. Le schéma ci-dessous illustre le principe. La station de travail et le RPI sont connectés au réseau de l'Université. Une fois la connexion *SSH* établie (nous verrons comment plus tard), il est possible de travailler sur la machine distante (ici le

RPI). Par exemple, la connexion redirige les informations des périphériques comme le clavier à la machine distante.



Pour effectuer la connexion à distance, nous vous donnerons un fichier contenant la clé RSA nécessaire à votre authentification sur les machines (<https://doc.fedoraproject.org/wiki/SSH : Authentification par clé RSA>, [https://fr.wikipedia.org/wiki/Chiffrement RSA](https://fr.wikipedia.org/wiki/Chiffrement_RSA))²). Il faudra la conserver pendant tout le projet.

Plus d'information sur le Raspberry PI ici : https://fr.wikipedia.org/wiki/Raspberry_Pi

4.4 JupyterLab

Le JupyterLab est un portail qui vous permet d'utiliser une fenêtre Terminal Linux et de programmer votre projet. La Figure 8 montre la page d'accueil du portail à l'adresse <http://sasdie.irisa.fr>.



Figure 8 : Page d'accueil du JupyterLab.

2 Cette clé est confidentielle. Elle est associée à votre compte et doit rester privée.

Une vidéo de présentation est ici : <https://youtu.be/PRSdCj51qyU>

Remarque : Les activités du JupyterLab sont surveillées. Il y a une limite de **60 sessions** simultanées pour des motifs d'espace mémoire. Il est possible qu'à un instant donné, vous ne puissiez pas vous connecter (surtout les derniers jours de rendu de TP).

5 Les étapes de travail (*sprints*)

Cette partie du document décrit l'ensemble des sprints que vous allez devoir effectuer.

Le projet se divise en 10 sprints :

1. Sprint 1 : Analyse du travail à effectuer
2. Sprint 2 : L'utilisation de JupyterLab
3. Sprint 3 : Le Shell
4. Sprint 4 : Utiliser le Raspberry PI
5. Sprint 5 : L'édition de fichiers textes sur le RPI
6. Sprint 6 : Le langage Python
7. Sprint 7 : La création d'un fichier CSV avec les données temps réels (JSON)
8. Sprint 8 : Le langage HTML
9. Sprint 9 : La production de pages HTML
10. Sprint 10 (Facultatif) : Calcul de pollution de pistes cyclables

Chaque sprint se compose des sections suivantes :

1. Tâches
2. Rendus du sprint
3. Creuser la question

Les éléments qui vous sont fournis ne constituent qu'une base d'explication, n'hésitez pas à faire des recherches sur Internet pour trouver des documents supplémentaires.

5.1 Sprint 1 : Analyse du travail à effectuer

Au préalable vous devrez connaître vos identifiants ENT pour vous connecter sur la station de travail.

Le cœur de ce Sprint consiste à prendre connaissance de l'ensemble du travail à faire et à tester le rendu de vos fichiers.

5.1.1 Tâches

Voici les tâches à effectuer pour ce premier sprint :

- 1) Lecture de la documentation ;
- 2) test du site Web de rendu de travail.

5.1.1.1 *Lecture de la documentation*

Le document complet à lire (celui-ci) est disponible à l'adresse :

<http://prototypel1.irisa.fr/documentation/documentation-projet.pdf>

Vous devez lire attentivement l'ensemble des consignes et les objectifs du travail demandé.

5.1.1.2 *Test du site Web de rendu de travail*

- 1) Récupérer le fichier « sprint1.txt » via l'adresse :
<http://prototypel1.irisa.fr/divers/sprint1.txt>
- 2) Ensuite, le publier sur le serveur de rendu via l'adresse :
<http://depot-l1miee.irisa.fr/webApps/RenduSAS/>

La vidéo est ici : <https://youtu.be/zPjoie69Q7o>

5.1.2 Rendus du sprint 1

1. Vous devez avoir fait un test de connexion sur le site de rendu de fichier (<http://depot-l1miee.irisa.fr/webApps/RenduSAS/>) avec le fichier *sprint1.txt*.

Lors des rendus, il est impératif de respecter le nom des fichiers, sans cela, nous ne pourrons pas vérifier que le travail a effectivement été effectué. Si vous avez un doute, n'hésitez pas à reposter le fichier.

5.2 Sprint 2 (JupyterLab) : L'utilisation de JupyterLab

Le portail JupyterLab vous permet de disposer d'une machine virtuelle où vous pourrez effectuer la partie de programmation Python ainsi que les travaux sur les commandes Shell. Ce portail vous permettra aussi de vous connecter sur les Raspberry PI.

5.2.1 Tâches

- 1) se connecter sur votre session du JupyterLab ;
- 2) télécharger sur votre station de travail / PC le fichier README.md ;
- 3) téléverser sur votre session du JupyterLab le fichier « sprint1.txt » du sprint précédent ;
- 4) ouvrir un notebook et exécuter un code python.

5.2.1.1 Ouvrir un notebook et exécuter un code python

Pour cet exercice vous devez :

- 1) créer un notebook ;
- 2) renommer celui-ci sasdie.ipynb ;
- 3) ajouter le programme ci-dessous dans la cellule d'exécution ;
- 4) modifier les coordonnées GPS ;
- 5) exécuter le programme.

```
import sasdie
sasdie.init()
c = sasdie.Sasdie()
c.setLogin("votre email")
c.setPasswd("votre numéro d'étudiants")
if c.connect() == False:
[tab]print ("la connexion à échoué")
[tab]exit(1)
#Mettre ici les coordonnées GPS (au format degrés décimaux) de vos
#bars ou restaurants préférés (au moins 1)
if not c.publierCoordonneesGPS("longitude0","latitude0"):
[tab]print("Erreur: position non postée !")
#recopier la ligne précédente pour
#ajouter de nouveaux bars ou restaurants
#et mettre les nouvelles coordonnées
if not c.publierCoordonneesGPS("longitude1","latitude1"):
[tab]print("Erreur: position non postée !")
```

Cet exercice permet de créer, par *crowdsourcing*, un fichier de coordonnées GPS que nous utiliserons dans l'un des sprints. Les données sont conservées de manière anonyme. **Attention de respecter l'indentation indiquée ci-dessus (la séquence [tab] est à remplacer par un caractère tabulation).**

L'exécution est correcte s'il ne s'affiche pas de message d'erreur.

5.2.2 Rendus du sprint 2

- 1) rendre le fichier README.md que vous avez téléchargé à partir de votre session JupyterLab :
- 2) rendre le fichier sasdie.ipynb

5.2.3 Creuser la question

Le logiciel Jupyter est maintenant très populaire. Vous trouverez des informations complémentaires ici : <https://jupyter.org/>. Il existe de nombreux tutoriels sur Youtube.

5.3 Sprint 3 (JupyterLab) : Le Shell

Ce sprint va vous permettre de découvrir l'application Terminal (dans l'environnement JupyterLab) qui donne accès à un interpréteur de commandes Shell. Un grand nombre d'outils informatiques s'utilise en « ligne de commande ». C'est l'une des principales interfaces de gestion.

Le Shell est un interpréteur de commande accessible à partir de l'application terminal (ou encore appelée console). Une illustration est en Figure 9. Le Shell permet d'accéder aux fonctionnalités du système d'exploitation. Il se présente sous la forme d'une invite de commande (aussi nommée « prompt », sur la Figure c'est « paralafac :~bodin\$ » qui permet de taper une commande. La commande est exécutée lorsque l'on entre un « retour chariot (↵) ». Ref. https://fr.wikipedia.org/wiki/Shell_Unix.

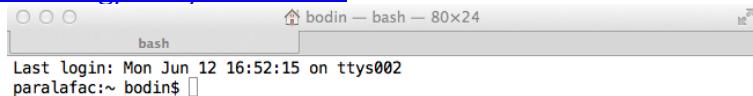


Figure 9 : exemple de fenêtre de l'application Terminal

Pour des explications complémentaires, vous pouvez consulter le lien suivant :
<https://doc.ubuntu-fr.org/terminal>

5.3.1 Tâches

Voici les tâches à effectuer pour ce sprint :

- 1) Comprendre et utiliser la commande Shell «pwd»
- 2) Comprendre et utiliser la commande Shell «ls»
- 3) Comprendre et utiliser la commande Shell «cd»
- 4) Comprendre et utiliser la commande Shell «cp»
- 5) Comprendre et utiliser la commande Shell «more»

6) Comprendre et utiliser la commande Shell «script»

5.3.1.1 La commande Shell «pwd»

La commande «**pwd**» (print working directory) indique le **répertoire courant**, c'est-à-dire le répertoire où l'on se trouve à un instant donné pendant la navigation dans le système de fichier. Il est noté « **.** ». Les commandes manipulant des fichiers s'appliquent par défaut à partir de ce répertoire.

Un **répertoire** (ou encore *dossier, directory*) désigne un emplacement dans un ordinateur correspondant à un ensemble de fichiers et d'autres répertoires. Les répertoires sont organisés en une structure d'arbre. La racine est le répertoire « **/** ». Votre répertoire personnel qui contient les fichiers vous appartenant a un alias simplifié « **~** » dans Linux.

Ref. [https://fr.wikipedia.org/wiki/Répertoire_\(informatique\)](https://fr.wikipedia.org/wiki/Répertoire_(informatique)),
https://fr.wikipedia.org/wiki/R%C3%A9pertoire_personnel

Voir la vidéo suivante pour une courte explication concernant l'organisation des fichiers : <https://youtu.be/1B5Rf09zTYY>.

La vidéo de démonstration de la commande « **pwd** » est ici :
<https://youtu.be/2CG-qLOegYk>

5.3.1.2 La commande Shell «ls»

La commande Shell «**ls**» (listing) donne la liste des fichiers dans un répertoire. Voici deux cas d'utilisation typiques :

1) Commande sans paramètre

\$ **ls** ↵

Donne la liste des fichiers du répertoire courant (celui indiqué par **pwd**)

2) \$ **ls** /tmp ↵

Donne la liste des fichiers correspondant au répertoire indiqué après **ls** (ici **/tmp**).

La vidéo de démonstration est ici : <https://youtu.be/z-VFRIjkrhU>

Remarque : la commande **ls -l** affiche les détails des fichiers dont l'heure de modification (ce qui est utile pour vérifier si la modification d'un fichier a bien eu lieu)

5.3.1.3 La commande Shell «more»

La commande Shell «**more**» affiche le contenu d'un fichier. Par exemple :

```
$ more sprint1.txt ↵
```

affiche le contenu du fichier « `sprint1.txt` ». Lorsque le contenu dépasse la taille de la fenêtre, la barre d'espace permet de passer à la suite du contenu. Si le contenu du fichier est composé de code binaire autre que ceux correspondants à des caractères ASCII, le contenu affiché est illisible.

La vidéo de démonstration est ici : <https://youtu.be/R47eJ2BUeyA>

5.3.1.4 La commande Shell « `cd` »

La commande Shell « `cd` » (change directory) permet de changer le répertoire courant. Voici deux cas d'utilisation possibles.

Commande sans paramètre

```
$ cd ↵
```

Positionne le répertoire courant sur le répertoire personnel de l'utilisateur (à chaque compte utilisateur correspond un ensemble de fichiers qui lui est propre).

```
$ cd /tmp ↵
```

Positionne le répertoire courant sur celui donné en paramètre (ici `/tmp`).

La vidéo de démonstration est ici : <https://youtu.be/LfrhC1Q1R9w>

5.3.1.5 La commande Shell « `cp` »

La commande Shell « `cp` » (copy) permet de copier un fichier. Par exemple :

```
$ cp sprint1.txt sprint2.txt ↵
```

Créera une copie du fichier `sprint1.txt` dans le répertoire courant. Dans ce cas, le fichier `sprint1.txt` doit se trouver dans le répertoire courant.

```
$ cp sprint1.txt /tmp/sprint2.txt ↵
```

Créera une copie du fichier `sprint1.txt` dans le répertoire `/tmp`.

La vidéo de démonstration est ici : <https://youtu.be/IImjd9Z5zRQ>

5.3.1.6 La commande Shell « `ping` »

La commande Shell « `ping` » interroge une machine connectée à l'Internet afin de savoir si celle-ci est accessible. Par exemple :

```
$ ping www.irisa.fr ↵
```

essaie de contacter la machine en charge du site web de l'Irisa.

La vidéo de démonstration est ici : <https://youtu.be/PKtbECoshps>

5.3.1.7 La commande Shell « script»

La commande Shell «script» permet d'enregistrer dans un fichier (par défaut *typescript*) tous les textes apparaissant à l'écran de la fenêtre de l'application terminal. L'enregistrement du texte est stoppé par une combinaison de deux touches « contrôle » et « d » (notée ^d). Par exemple :

```
$ script ↵
$ ls ↵
xxxxx
yyyyy
www
$ ^d
```

produira un fichier *typescript* contenant le texte suivant :

```
$ ls ↵
xxxxx
yyyyy
www
```

La vidéo de démonstration est ici : <https://youtu.be/JRVXc76fWSA>

5.3.2 Rendus du sprint 3

Pour ce sprint vous devez produire le fichier *typescript* construit par suite de commande Shell :

```
script
cd
pwd
ls
^d
```

et ensuite rendre le fichier *typescript*.

5.3.3 Quelques trucs utiles

- La commande shell "history" permet d'afficher les commandes Shell précédentes ;

- pour compléter un nom de fichier dans une commande Shell utilisez la touche tabulation ;
- pour voir les possibilités de fichiers dans une commande Shell taper deux fois de suite sur la touche tabulation ;
- la commande «*man nom_de_la_commande*» (par exemple *man ls*) affiche la documentation des commandes du Shell. Le terme « *man* » fait ici référence à « *manual* ».

Ne pas hésiter à rechercher sur google/duckduckgo/... des informations sur les messages d'erreur et des compléments d'information sur le Shell (p.ex. les messages d'erreurs de la commande *ssh*).

5.3.4 Autres règles importantes

- Ne pas écrire des commandes ou des instructions de programme que vous ne comprenez pas ;
- lire et relire la documentation (RTFM - Read The Fucking Manual ; [https://fr.wikipedia.org/wiki/RTFM_\(expression\)](https://fr.wikipedia.org/wiki/RTFM_(expression))) ;
- prenez des notes.

5.3.5 Creuser la question

On trouve de nombreuses ressources à propos des commandes Shell, voici quelques exemples :

- 1) <http://www.tuteurs.ens.fr/unix/shell/>
- 2) <https://doc.ubuntu-fr.org/permissions>

5.4 Sprint 4 (RPI): Utiliser le Raspberry PI

Dans ce sprint vous allez commencer à utiliser le Raspberry PI (RPI). C'est lui qui récupère les données de capteurs en temps réel. Les Raspberry PI sont accessibles via Internet.

5.4.1 Tâches

Les tâches à effectuer pour ce Sprint sont de :

- 1) tester l'accès au RPI ;
- 2) se connecter sur le RPI avec la commande Shell «*ssh*» ;
- 3) copier des fichiers entre le JupyterLab et le RPI avec la commande Shell « *scp*».

5.4.1.1 Tester l'accès aux RPI

Son nom sur le réseau est «**pilic[Y].irisa.fr**». [Y] est le numéro du RPI. Chaque RPI possède son propre numéro (Par exemple *pilic19.irisa.fr* désigne le RPI 19). L'information concernant celui que vous devrez utiliser vous sera envoyée par mail.

Pour vérifier que celui-ci est bien en route, dans l'application Terminal taper la commande suivante :

```
$ ping nom_du_raspberry_PI ↵
```

5.4.1.2 Se connecter sur le RPI avec SSH

Pour vous connecter, vous devez disposer d'une clé «rsa» correspondant à votre compte utilisateur sur les RPI. Cette clé est un fichier qui contient des informations de connexion. La clé vous a été distribuée, vous avez dû la copier à la racine de votre session JupyterLab. Les commandes Shell (dans l'application Terminal) pour se connecter sont alors :

```
$ cd ↵ (se mettre à la racine de votre compte)  
$ ls ↵ (vérifier que la clé rsa se trouve bien dans le répertoire)  
$ ssh pilicX@.....fr -i cle_rsa ↵
```

La vidéo de démonstration se trouve ici : <https://youtu.be/WRw6dm2W9UM>

5.4.1.3 Faire des copies de fichiers entre le JupyterLab et le RPI

La commande «scp» permet de copier des fichiers entre le RPI et la station de travail/JupyterLab. Par exemple dans la fenêtre terminal du JupyterLab :

```
$ scp -i cle_rsa sprint1.txt pilicX@pilicY.irisa.fr:.. ↵
```

copie le fichier `sprint1.txt` dans votre répertoire personnel (c'est le texte «`..`» qui indique cela) de l'utilisateur `pilicX` sur le RPI `pilicY.irisa.fr` (ou `pilicX@pilicY.irisa.fr`). La clé RSA est indiquée après l'option `-i` qui doit alors être placée au début de la commande.

La vidéo de démonstration se trouve ici : <https://youtu.be/HyV92qqPuYY>

5.4.1.4 Tâche

- 1) Copier le fichier «`monfichier.txt`» sur le RPI en utilisant la commande `scp` ;
- 2) déplacer ce fichier dans le répertoire `2022` du RPI ;
- 3) exécuter le programme Python `traiterMonfichier.py` ;
- 4) recopier le nouveau fichier «`monfichier.txt`» sur la session JupyterLab.

5.4.2 Rendus du sprint 4

Télécharger le fichier `monfichier.txt`, via l'interface de rendu de TP.

5.4.3 Creuser la question

Plus d'information à propos de SSH : <https://doc.ubuntu-fr.org/ssh>

5.5 Sprint 5 (RPI) : L'édition de fichier texte sur le RPI

L'utilisation d'emacs sur le RPI à travers la fenêtre terminal du JupyterLab ne fonctionne pas correctement. Vous remplacerez emacs par le logiciel nano sur le RPI.

Dans ce sprint, il s'agit de se familiariser avec un éditeur de texte (à ne pas confondre avec Microsoft Word qui est un logiciel de traitement de texte). Ici, l'éditeur de texte permet de saisir le code des programmes ou de manipuler des fichiers CSV.

Les fichiers CSV (Comma-Separated Values) servent essentiellement au stockage de données tabulées en format texte. Ils sont organisés de manière à ce que chaque ligne de texte corresponde à une ligne de tableau, les virgules (ou points-virgules) séparent les colonnes. Ce format est particulièrement adapté au stockage de données de capteurs.

L'éditeur de texte que nous allons utiliser est «`emacs`» (et/ou `nano`). Emacs est très ancien, puissant et disponible sur de nombreuses machines (son inventeur, Richard Stallman est sur la Figure 10). Beaucoup d'applications réutilisent les raccourcis d'Emacs.

Pour lancer `emacs` la commande est :

```
$ emacs dataBAM.csv
```

Quelques commandes de l'éditeur `emacs` sont les suivantes :

<code>^X^f</code>	contrôle-x contrôle-f, chargement d'un fichier
<code>^X^s</code>	contrôle-x contrôle-s, écriture du fichier
<code>^X^c</code>	contrôle-x contrôle-c, quitter emacs
<code>↑</code>	Déplacement du curseur à la ligne au-dessus
<code>↓</code>	Déplacement du curseur à la ligne au-dessous
<code>⇒</code>	Déplacement du curseur d'un caractère vers la droite
<code>⇐</code>	Déplacement du curseur d'un caractère vers la gauche

Notez que sur Jupyter, certains raccourcis peuvent être interprétés par le navigateur avant `emacs` (fermeture de fenêtre, ...). Préférer dans ce cas les raccourcis commençant par « escape » (touche « Echap »).

Richard Stallman initiateur du projet
GNU (GNU's Not UNIX) est
l'inventeur d'emacs.



Figure 10 : Richard Stallman, l'initiateur du projet GNU.

La vidéo de démonstration se trouve ici : <https://youtu.be/ebFOiqsilxo>

5.5.1 Tâches

Voici les tâches à effectuer pour ce Sprint :

- 1) créer un fichier «`monprogramme.py`» avec `emacs` ;
- 2) ouvrir avec `emacs` un fichier existant (à la racine du compte sur le RPI «`dataBAM.csv`») et le modifier. Ce fichier contient des mesures de PM produites par la station Laennec d'AirBreizh.

Pour ce sprint, vous devez être connecté sur le RPI comme vu au sprint précédent.

5.5.1.1 *Créer un fichier monprogramme.py avec emacs*

Pour créer un fichier avec `emacs`, il vous suffit de taper la commande dans l'application Terminal.

- 1) Mettre le texte «`print("ceci est un essai")`» ;
- 2) sauvegarder le fichier `monprogramme.py` ;
- 3) récupérer le fichier sur la station de travail (avec la commande `scp`).

5.5.1.2 *Modifier un fichier existant*

- 1) Dans le répertoire `/home/pilicX/2022` (`/home/pilicX` est le répertoire personnel de l'utilisateur pilicX), éditez le fichier «`dataBAM.csv`» et remplacez tous les «`/21`» des dates par des «`/2021`». Pour cet exercice vous devrez rechercher la commande «`trouver-replacer`» d'`emacs` ;
- 2) sauvegardez le fichier ;
- 3) récupérez le fichier sur la session JupyterLab (avec la commande `scp`).

5.5.2 Rendus du sprint 5

Pour ce sprint vous devez rendre via la page de rendu :

- 1) Le fichier «`monprogramme.py`» ;
- 2) le fichier «`dataBAM.csv`».

5.5.3 Creuser la question

La liste des raccourcis d'`emacs` :

<https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf>

Un tutoriel : <https://www.tuteurs.ens.fr/unix/editeurs/emacs.html>

La documentation complète d'`emacs` :

https://www.gnu.org/software/emacs/manual/html_node/emacs/index.html

La documentation Nano (ou ^G dans l'éditeur) :

Officielle: <https://www.nano-editor.org/docs.php>

En Français : <https://linuxpedia.fr/doku.php/commande/nano>

5.6 Sprint 6 (JupyterLab) : Le langage Python

Dans ce sprint, vous allez commencer à travailler avec le langage Python. Ce langage est celui qui va être utilisé pour mettre en œuvre, entre autres, la génération des pages Web. Le Python est un langage interprété.

Un **langage interprété** est un langage de programmation dont les programmes sources sont directement exécutés par un logiciel appelé *interpréteur*. Il se distingue des langages compilés qui eux doivent d'abord être traduits en instructions machines binaires avant de pouvoir être exécutés. Il existe de nombreux langages interprétés : Python, Perl, Javascript, Ruby, Lua, ... On nomme souvent un programme qui sera interprété comme un *script*.

Ref. [https://fr.wikipedia.org/wiki/Interprète_\(informatique\)](https://fr.wikipedia.org/wiki/Interprète_(informatique))

Pour démarrer la programmation en langage Python, vous trouverez des exemples dans le répertoire `ExemplesPython` sur le RPI et JupyterLab («`cd ; cd ExemplesPython`»).

Pour tester ces programmes, utilisez la commande `python` (dans un Terminal Shell) suivie du fichier de code à exécuter. Par exemple :

```
$ python print.py ↵
```

Pour examiner le contenu du fichier, vous pouvez utiliser soit «`emacs`» soit la commande Shell «`more`» :

```
$ more print.py ↵
```

L'ensemble de ces programmes comporte les constructions nécessaires pour le travail demandé. Vous devez connaître le fonctionnement des éléments suivants :

- L'instruction `print`.
- Les variables.
- Les expressions.
- Les conditionnelles.
- Les boucles.
- Les chaînes de caractères.
- Les listes.
- L'utilisation d'API.

Il existe de nombreux tutoriels disponibles, vous pouvez considérer ceux-ci en priorité :

- https://python.sdv.univ-paris-diderot.fr/00_avant_propos/
- <https://pythontutor.com/>
- https://koor.fr/Python/Tutorial/python_introduction.wp

Dans ces tutoriels concentrez-vous sur les exemples Python du répertoire ExemplesPython, ces exemples contiennent l'ensemble des constructions dont vous avez besoin.

Les instructions Python que vous utiliserez devront être les plus simples possibles. Évitez les copier-coller issus d'Internet.

5.6.1 Tâches

Vous devez :

1. écrire un programme qui affiche à l'écran «hello world !» ;
2. écrire un programme qui concatène trois chaînes de caractères ;
3. écrire un programme qui additionne 123456 et 876543 ;
4. écrire un programme qui affiche, à l'aide d'une boucle, les nombres de 3 à 25, et indique si le chiffre est pair ou impair ;
5. écrire un programme qui remplit une liste de chaînes de caractères et concatène une même chaîne à celles de la liste ;
6. écrire un programme utilisant une API.

5.6.1.1 Hello world

Écrire un programme qui affiche à l'écran « *hello world !* ». Notez que c'est une sorte de tradition de l'informatique de commencer par un exemple de ce type : https://fr.wikipedia.org/wiki/Hello_world

Le programme sera dans le fichier « `hello.py` »;

5.6.1.2 Concaténation de chaînes de caractères

Écrire un programme qui concatène les trois chaînes de caractères :

1. «exercice de»
2. «concaténation»
3. «de chaînes»

et affiche le résultat à l'écran. Le programme sera dans le fichier « `concat.py` ».

5.6.1.3 Additions

Écrire un programme qui additionne 123456 et 876543 et affiche le résultat à l'écran. Le programme sera dans le fichier « `add.py` »;

5.6.1.4 Boucle

Dans cet exercice vous allez écrire un programme qui énumère les entiers de 3 à 25 (compris), affiche le chiffre et indique si celui-ci est pair ou impair.

[tab] indique l'insertion d'une tabulation ou d'un ensemble fixe d'espace, elle fait partie du langage Python pour indiquer les instructions incluses dans un bloc.

La structure du code devra ressembler à :

```
for ... :  
[tab] if ... :  
[tab][tab] ...  
[tab] else :  
[tab][tab] ...
```

Le programme sera dans le fichier « **boucle.py** ».

5.6.1.5 Liste

Cet exercice comporte deux parties :

1. créer une liste avec 5 chaînes de caractères (de votre choix) ;
2. écrire une boucle qui parcourt la liste et ajoute la chaîne « extension de chaîne » à chaque élément de la liste ;
3. écrire une nouvelle boucle qui affiche le résultat.

La structure du code devra ressembler à :

```
# initialisation liste  
T=[]  
...  
# boucle  
for ... :  
[tab] T[...] = T[...] ...  
  
# boucle d'affichage des résultats  
for ... :  
[tab] ...
```

Le programme sera dans le fichier « **liste.py** ».

5.6.2 Utilisation de l'API d'accès aux données

Dans ce sprint, vous allez utiliser une API, dans un programme Python, pour communiquer avec le serveur Web. L'API comporte les instructions à corriger ou à compléter suivantes :

```
#Initialisation et importation des fonctions nécessaires  
import sasdie  
#Quelques initialisations de variables
```

```
sasdie.init()  
#Création de la structure qui fournit les méthodes  
c = sasdie.Sasdie()
```

```
#Connexion au serveur « prototypell.irisa.fr »  
c.setLogin(email)  
c.setPasswd(numéroetudiant)  
if not c.connect():  
    print("La connexion a échouée")  
    exit(1) # arrêt du programme
```

```
#Récupération de la clé de votre compte sur  
#« prototypell.irisa.fr »  
c.macle()
```

```
#Publier un contenu sur le serveur  
# texte page est une chaîne de caractères  
c.publierpage_html(texte page)
```

Pour cette tâche vous devez écrire le code qui récupère la clé de votre compte sur « prototypell.irisa.fr » et la publie ensuite comme texte de la page.

Le programme sera dans le fichier « API.py ».

5.6.3 Rendus du sprint 6

Pour ce sprint, vous devez rendre les fichiers suivants :

- 1) le fichier « hello.py » ;
- 2) le fichier « concat.py » ;
- 3) le fichier « add.py » ;
- 4) le fichier « boucle.py » ;
- 5) le fichier « liste.py » ;
- 6) le fichier « API.py » .

Si vous avez créé ces programmes avec un notebook, vous pouvez obtenir le fichier demandé en ajoutant à la fin de votre programme la commande

```
c.exportSource("nomdunotebook.ipynb", "nomdufichierarendre.py")
```

Cette fonction publie automatiquement le fichier sur le site de rendu (**pensez à sauvegarder le notebook avant**). Sinon utiliser l'interface de rendu.

5.6.4 Creuser la question

Quelques références pour creuser la question :

- 1) https://fr.wikipedia.org/wiki/Hello_world
- 2) <http://web.univ-ubs.fr/lmba/lardjane/python/c4.pdf>

- 3) <https://docs.python.org/fr/3/tutorial/>
- 4) <https://docs.python.org/fr/3/c-api/index.html>
- 5) https://www.youtube.com/watch?v=CXf_7W-qACU (APPRENDRE PYTHON, les bases pour les débutants)

5.7 Sprint 7 (RPI) : La création d'un fichier CSV avec les données temps réel

Le répertoire de travail pour ce sprint est `~/2022`.

Il s'agit de produire un fichier CSV contenant les 10 dernières valeurs de capteur produites par le capteur fixe de la Figure 3. Les données sont produites de manière asynchrone (i.e. qui n'est pas synchronisée avec la demande d'accès à une mesure) environ toutes les 20 secondes. Le programme devra être exécuté sur le RPI. Pour ce travail vous disposez des méthodes python suivantes :

```
#Importe les méthodes nécessaires  
import RPI_Sasdie_Lib
```

```
# Crée la structure de données (classe) pour l'accès aux méthodes  
d = RPI_Sasdie_Lib.DataStream()
```

```
#Lit la dernière valeur produite par le capteur.  
#Attention à chaque fois que vous appelez la méthode,  
#elle ne produit pas nécessairement une nouvelle valeur. La valeur  
#retournée (data) comporte deux champs : data.count le numéro de  
#l'échantillon produit et data.pm25 la mesure des PM25.  
  
data = d.lectureDonnéesCourante()
```

```
#Effectue l'écriture du contenu de contenuFichier  
#dans le fichier data_pm25.csv.  
#contenuFichier est une chaîne de caractères. A chaque appel de  
#cette méthode l'ensemble du contenu du fichier est réécrit. Il  
#n'y a PAS de concaténation du nouveau contenu à la fin du fichier.  
  
d.ecritureDuFichierCSV(contenuFichier)
```

5.7.1 Tâche

Écrire le programme (`lectureCapteurFixe.py`) qui collecte exactement 10 données de capteurs et les enregistre dans un fichier CSV (`data_pm25.csv`) comportant deux colonnes :

- 1) le numéro de l'échantillon de 1 à 10 ;
- 2) la valeur de PM25 produite par le capteur.

La première ligne du fichier CSV sera «Numéro d'échantillon;PM25» (on utilisera donc le « ; » comme séparateur de colonnes).

Pour cette exercice vous ne devez pas utiliser la commande Python « `time.sleep(...)` »

5.7.2 Rendus du sprint 7

Pour ce sprint, vous devez rendre les fichiers `lectureCapteurFixe.py` et `data_pm25.csv`.

5.8 Sprint 8 (JupyterLab): Le langage HTML

Le langage HTML est un langage pour produire des pages HTML. Le langage est fondé sur la notion de balise ayant la syntaxe suivante :

- `<balise> texte entre les balises de début de fin </balise>`

La « `<balise>` » indique le traitement qui doit être appliqué au texte entre les balises (de début et de fin) pour son affichage par le navigateur web (p. ex. Firefox). C'est lui qui comprend ce langage. La chaîne « `</balise>` » est la balise de fin. Par exemple pour afficher du texte en gras on écrira :

- ` texte à mettre en gras (bold) `

Les vidéos suivantes donnent des explications complémentaires :

- 1) créer une page simple : https://youtu.be/HEo_oA7C7o ;
- 2) créer des liens : https://youtu.be/nxaCLuD_X7A ;
- 3) mettre un tableau dans une page : <https://youtu.be/eiMLc7q3340>.

Les exemples de page HTML sont disponibles sur le Jupyter ou le Raspberry PI dans le répertoire `ExemplesHTML`. De nombreux exemples sont disponibles sur Internet.

Pour ce TP vous testerez les pages (avec Firefox) sur la station de travail, mais la création des pages se fera dans JupyterLab.

Le langage HTML est celui que vous allez utiliser pour afficher les données CSV collectées avec le RPI.

5.8.1 Tâches

Voici les tâches à effectuer pour ce sprint :

- 1) écrire un code Python (fichier `genTabDataHTML.py`) qui produit la page HTML avec un tableau contenant les données du capteur du fichier produit sur le RPI (`data_pm25.csv`).

Voici une reproduction du résultat :

Données RPI

Numéro d'échantillon	PM25
1	10.12
2	10.12
3	10.12
4	10.12
5	10.12
6	10.12
7	10.12
8	10.12

Pour ce travail vous disposez des méthodes Python suivantes :

```
#lecture du fichier data_pm25.csv
#La variable donnees peut typiquement être parcourue par une boucle
#« for row in donnees: ». L'expression row[0] correspond à l'élément
#en première colonne et row[1] de celui en deuxième colonne)

donnees = c.lireDonneesPollutionRPI()
```

```
#Publication du contenu HTML, "textHTML"
#est une chaîne de caractères
#correspondant au source de la page.

c.serveur.publierpage_html("textHTML")
```

```
#Si vous voulez faire le développement dans un notebook
#la méthode d'export sur le site de rendu est la suivante

c.exportSource("nom-du-notebook.ipynb","genTabDataHTML.py")
```

Ces méthodes sont disponibles avec l'API d'accès aux données du sprint 6.

5.8.2 Rendus du sprint 8

Pour ce sprint, vous devez rendre le fichier « *genTabDataHTML.py* » via l'interface de rendu (ou l'export du notebook).

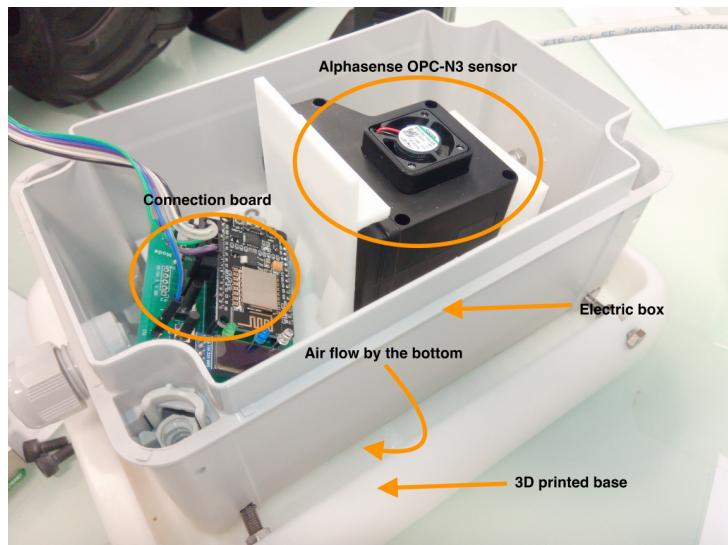
5.8.3 Creuser la question

Vous pouvez creuser la question en consultant les deux sites suivants :

- 1) <http://www.delprat.org/cours/index.htm> (une introduction en français) ;
- 2) <https://www.w3schools.com/html/> (en anglais).

5.9 Sprint 9 (JupyterLab) : Génération d'une carte HTML

Le fichier CSV `data-pm-sasdie-bus.csv` contient des relevés de qualité de l'air dans la métropole rennaise. Les données ont été collectées par des bus équipés de capteur AlphaSence OPC-N3 (Figure 11).



Le fichier comporte des lignes avec les champs suivants

- * Les coordonnées GPS, capturées par la station de mesure (mobile), du Nord-Ouest au Sud-Est, longitude puis latitude:
vblon: longitude de départ
vblat: latitude de départ
velon: longitude de fin
velat: latitude de fin
- * La période de mesure en millisecondes et au format POSIX UTC:
start_date: début de mesure
stop_date: fin de mesure
- * La valeur mesurée:
value: PM2.5 en µg/m³
- * Le nom de la station mobile et du capteur (n°série de l'OPC-N3);
station: nom de ma station de mesure
sensors: nom du capteur
- * La période de mesure en mode texte.
vbdate: measure starts.
vedate: measure stops.

Un exemple de ligne est (ici découpé en plusieurs lignes) :

```
-1.684122709;48.129878342;-1.683787022;48.129061521;  
1604994820298;1604994820298;  
4.289999961853027;  
"Bus700";
```

```
"OPC_N3:20";  
"2020-11-10T07:53:40.298Z";"2020-11-10T07:53:40.298Z";
```

Il n'y a pas une seule valeur pour la position GPS et la date de prise de la mesure car il existe une incertitude sur la position et le moment de prise de la mesure. Un intervalle est donc fourni. Pour ce travail vous devrez faire la moyenne arithmétique des valeurs.

Pour ce travail, **toujours avec l'API d'accès aux données du sprint 6**, vous disposez des fonctions suivantes après import de « `sasdie.py` » :

```
#lecture de data-pm-sasdie-bus.csv  
#et création de la liste des échantillons.  
#csvdp est une liste que vous pouvez parcourir avec une boucle for.  
csvdp = c.lireDonneesPollution("data-pm-sasdie-bus.csv")
```

```
#lecture des emplacements déclarés  
# avec la fonction publierCoordonneesGPS.  
# csvcs est une liste que vous pouvez parcourir avec une boucle for.  
csvcs = c.lireCoordonneesGPS()
```

```
#création de l'objet carte (c.macarte)  
c.creerCarte()
```

```
#ajout d'un marqueur sur la carte  
#"latitude","longitude" sont les paramètres chaînes de caractères  
# qui seront extraits des contenus des fichiers CSV (csvcs ou csvdp)  
c.macarte.ajoutCercleSurLaCarte("longitude","latitude",  
                                 diamètre_en_mètres,"étiquette")  
  
c.macarte.ajoutMarqueurSurLaCarte("longitude","latitude",  
                                   "étiquette")  
  
#En cas d'inversion de la latitude et de la longitude  
#vos marqueurs seront sans doute du côté de Madagascar
```

```
#produit le texte HTML (et javascript) de la carte  
cartehtml = c.macarte.produireHTMLCarte()
```

```
#Publication de la page sur le serveur prototypel1.irisa.fr  
c.serveur.publierpage_html("textpage")
```

5.9.1 Tâches

- 1) Écrire un code (`carte.py`) qui publie une carte de Rennes avec l'ensemble des points de mesures du fichier `data-pm-sasdie-bus.csv` placés sur la carte (par la méthode `ajoutCercleSurLaCarte`). Pour calculer les coordonnées du marqueur sur la carte vous ferez les moyennes arithmétiques des longitudes (`vblon` et `velon`) et latitudes (`vblat` et `velat`) de chaque échantillon. L'étiquette du marqueur sera la valeur de la mesure de pollution ;

- 2) ce code produira la moyenne des valeurs de pollution pour les PM25. Cette valeur sera utilisée comme étiquette d'un marqueur de 500 mètres de diamètre qui sera positionné place de la Mairie de Rennes ;
- 3) ajouter sur la carte les positions des bars et restaurants déclarés (`lireCoordonneesGPS()`) lors d'un précédent script (avec la méthode `ajoutMarkerSurLaCarte`).

5.9.2 Rendus du sprint 9

Le fichier contenant le code du programme Python `carte.py`. Vous rendrez ce fichier via l'interface (ou la fonction python d'export si dans un notebook).

5.9.3 Creuser la question

Si vous souhaitez découvrir des aspects supplémentaires sur la surveillance de la qualité de l'air : <http://aqmo.irisa.fr/fr/resultats/workshop-aqmo/>

5.10 Sprint 10 (JupyterLab): Exercice pistes cyclables + pollution

Ce travail utilisera aussi les données remontées par les bus de la plateforme AQMO pour calculer des pistes cyclables exposées à la pollution.

A l'aide d'un programme python, vous devez produire une carte qui comporte :

- 1) Les données remontées des bus
- 2) Les pistes cyclables (issues de l'open-data de Rennes Métropole)
- 3) Les pistes cyclables qui ont une intersection avec une zone ayant une moyenne de pollution supérieure à $10 \mu\text{g}/\text{m}^3$

Vous pouvez faire les développements sur le portail sasdie.irisa.fr. Pour les valeurs de pm vous les écrêterez à $40 \mu\text{g}/\text{m}^3$.

Normalement les bibliothèques dont vous avez besoin sont déjà disponibles. Si ce n'est pas le cas, vous pouvez demander l'ajout dans le JupyterLab.

5.10.1 Rendu du sprint

- 1) Un notebook comprenant les codes Python et les explications est à rendre et à envoyer à bodin@irisa.fr.
- 2) Faire une démonstration du programme à l'un des enseignants.

6 Résumé des rendus du projet et dates limites

Les dates de rendu seront précisées par messagerie suivant l'emploi du temps des groupes.

Rappel, pour chaque rendu de fichier via l'interface (ou la fonction Python)

<http://depot-l1miee.irisa.fr/webApps/RenduSAS/>

vous pouvez aussi envoyer un message avec comme contenu, le fichier en attachement, à l'adresse sasdie2019@gmail.com et avec comme sujet « SASDIE – sprint *numéro* – *titre du sprint correspondant – sujet*».

7 Conclusion

Le travail proposé dans ce document présente une vue transversale des technologies des systèmes informatiques. De nombreux concepts utilisés ici ne sont pas étudiés en profondeur, ce sera l'objet de la suite de vos études d'informatique et/ou d'électronique.

Il est important de retenir qu'au-delà des connaissances que vous devez acquérir, la gestion de votre travail et la manière d'aborder l'apprentissage sont des éléments essentiels de vos études universitaires.

Nous espérons que le projet que nous vous avons proposé a initialisé ce processus.

8 FAQ

LA FAQ est disponible ici :

<https://foad.univ-rennes1.fr/course/view.php?id=1010963>