

Outils Collaboratifs

TP(s)

Enoncés et Exercices

par Philippe KISLIN-DUVAL

Licence Informatique
Institut d'Enseignement à Distance

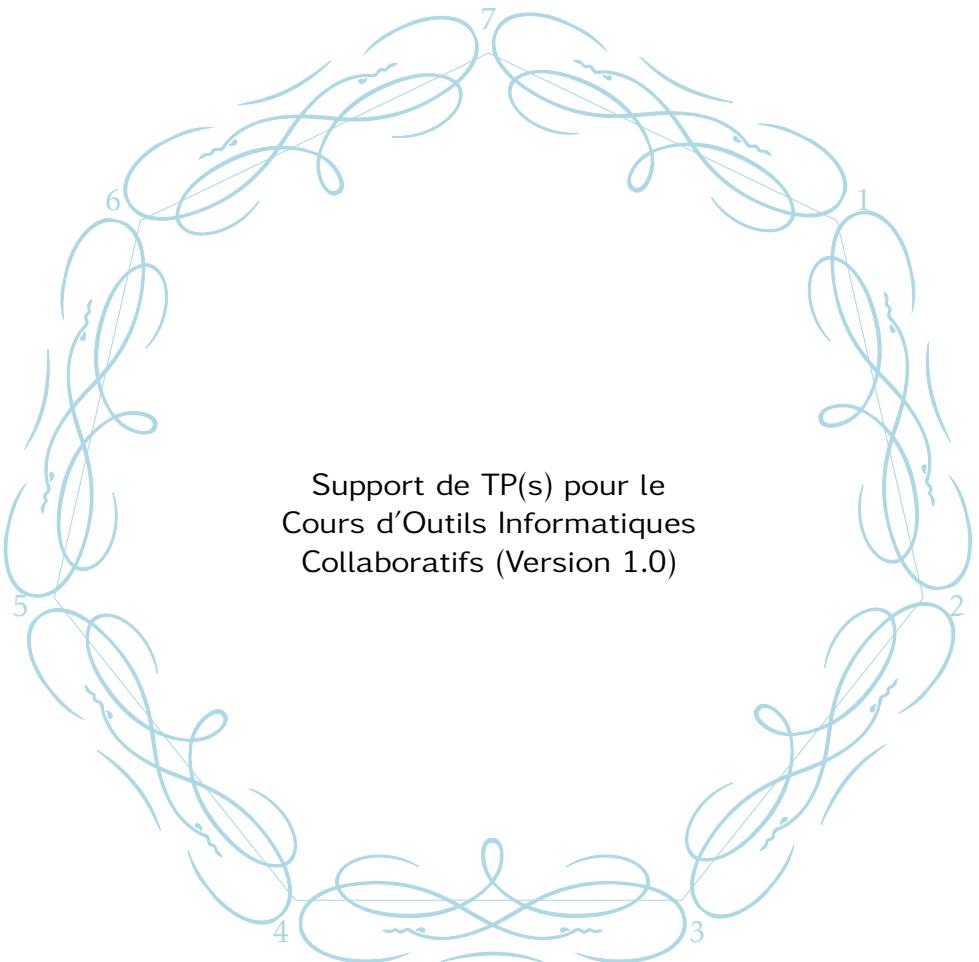


Mai 2021

Version 1.0 du 1er juin 2022

Copyleft - Free Documentation License(GFDL)

L
I
C
E
N
C
E



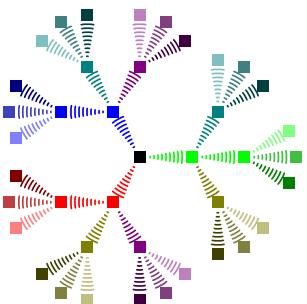
Support de TP(s) pour le
Cours d'Outils Informatiques
Collaboratifs (Version 1.0)

Sommaire

Introduction	6
Organisation du Cours et Présentation des Exercices	7
I TP 1 - Outils collaboratifs - Git et (Concepts) Couverts	18
1 Git : l'Outil Collaboratif du Développeur	19
1.1 Qu'est-ce que la Gestion de Versions?	20
1.2 Les Commandes de Base	22
1.3 Créer un Nouveau Projet	24
1.4 Cloner un Projet Existant	25
1.5 les Commits	25
1.6 Quelques Commandes Supplémentaires	31
1.7 Collaborer : Les Repositories Distants	33
1.8 Les Exercices du TP	38
1.9 Quelques Liens pour Aller plus Loin...	44
II TP 2 - Requêter à l'aide des API	46
2 API - Collecter de l'Information de Manière Collaborative	47
2.1 Qu'est-ce qu'une API?	47
2.2 S'Entraîner aux Actions avec une API	48
2.3 Utiliser l'API REST de Github	52
2.4 Un Appel API : une autre Approche	52
2.5 Requêter Automatiquement avec un Langage de Programmation : Python	55
2.6 Connaître les Limites de l'API	60
2.7 Construire une Représentation Graphique des Résultats	62

2.8	Découvrir une seconde API : Hacker News	70
2.9	Jouer avec une troisième API : Giphy	76
2.10	Les Exercices du TP	82
2.11	Quelques Liens pour Aller plus Loin...	85
III	TP 3 - Créer sa Propre API	86
3	Développer une RESTful API avec FastAPI	87
3.1	Qu'est-ce FastAPI?	87
3.2	Installation de FastAPI	88
3.3	Quelques Informations (Facultatives) sur le Traitement des E/S Asynchrones	93
3.4	Un Premier Exemple avec FastAPI	96
3.5	Aller plus Loin avec FastAPI	99
3.6	Un Squelette d'Application à Adapter	100
3.7	Les Exercices du TP	108
3.7.1	Compléments pour le Bonus : Implémenter une Fonction de Reconnaissance de Visages en Temps Réel . .	110
3.8	Quelques Articles et Livres pour Aller plus Loin...	126
IV	TP 4 - Médias Immersifs, Réalité Virtuelle et Partages Collaboratifs	129
4	Mondes Virtuels Immersifs et Plateforme Collaborative	130
4.1	Quelques Mots sur les Mondes Virtuels : Second Life, Fortnite, Roblox, Blockchain et Métavers(e)	131
4.2	Découverte de la Plateforme Collaborative Streamlit	134
4.2.1	Installation de Streamlit (Python >= 3.7)	135
4.2.2	Un Tout Premier Programme	137
4.3	Quelques Liens pour Aller plus Loin...	139
4.4	Les Exercices du TP	140
4.5	Et pour Clore ces Notes de Cours...	147

Objectifs et Organisation des TP(s)



Ces énoncés de TP(s) sont destinés aux étudiant.e.s de première année de la licence informatique de l'IED de Paris 8. Il est organisé, comme pour GIL, en quatre TP(s) (sans doute 5 dans la prochaine mouture). Comme pour les autres cours, avant de commencer la lecture de ces énoncés et notes introducives, veuillez prendre connaissance du **Petit Guide** introductif qui vous donnera toutes les indications de présentation et d'organisation du rendu de vos exercices sous la forme **d'un unique PDF** accompagné d'une **archive** comprenant vos fichiers et codes-sources. Comme indiqué sur le Moodle, n'oubliez pas de m'adresser, au préalable, **un mail pour m'indiquer votre intention de commencer ce cours** (ces TP(s)).

Petit Guide à Lire avant de Commencer

1 - Organisation Générale du Cours

La plupart des cours de la Licence Informatique de l'IED s'étale sur plusieurs semaines et demande d'accorder un temps plus ou moins long en travail et investissements personnels. Comptez environ **une semaine complète de travail** pour un chapitre, voire plus...

En général, le premier tiers des cours et exercices, est d'une d'approche relativement aisée : découverte des fondamentaux, applications simples. Le second tiers correspond à des notions plus approfondies et demande un temps plus important pour leurs assimilations et pour la réalisation des exercices. Le dernier tiers est, vous l'avez compris, le plus complexe. Tous les cours se terminent par **un projet** qui nécessite de réinvestir une grande partie de ce qui aura été vu et appris tout au long des différents chapitres.

Le projet quant à lui, véritable synthèse demande une très grande masse de travail : une réalisation pratique, le plus souvent sous la forme de codage informatique, mais également, et c'est le plus important, doit se structurer autour d'une explication méthodologique construite. C'est pour cela qu'il compte pour **presque la moitié** de la notation globale finale.

En raison du fort coefficient du projet, vous ne serez évalué(e) qu'après avoir terminé **l'intégralité du cours**, chapitre après chapitre, projet terminal inclus.

Pour toutes ces raisons, n'attendez pas les dernières semaines du mois d'août pour terminer vos exercices, pour vous lancer sur des projets particulièrement chronovores. De même, commencer un cours à moins de trois mois, voire moins, du rendu administratif des résultats est une gageure.

2 - Organisation Particulière des Correspondances

Avant de commencer le cours, vous devez **préalablement m'envoyer un premier mail** pour m'informer, comme indiqué sur la plateforme Moodle, de **votre intention de commencer le cours**, ou de continuer celui-ci si vous étalez son achèvement sur deux années universitaires. Il faudra cependant que vous réitériez cette demande pour chaque cours que vous suivrez, ce qui me permettra de m'organiser en créant pour celui-ci et en amont, un dossier à votre nom.

Pour tous les échanges que nous aurons (notifications, rendus d'exercices, questions, impressions, ...), je vous invite à utiliser l'adresse mail fournie sur la plateforme. De votre côté, n'hésitez pas à me communiquer (et à utiliser conjointement) une autre adresse mail que celle de l'Université dont les serveurs peuvent être quelquefois "surchargés".

Respectez bien **la forme de l'Objet** comme indiqué dans la page du LMS Moodle et rappelé ici :

IED - L(1..3) - NOM DU COURS - CHAPITRE (1..10) - Prénom Nom - ID

écrit en majuscules afin d'éviter que votre message ne se perde dans les quelques dizaines mails que je reçois quotidiennement. Il est vraiment important que cette forme soit utilisée car vos envois sont filtrés sur ces critères. J'essaie de suivre du mieux possible vos progressions et l'envoi de vos messages. Mais, comme la technologie des serveurs de messagerie a ses caprices (qui échappe quelquefois à la raison) et moi-même qui peux "oublier" un message parmi tant d'autres, n'hésitez pas à me renvoyer votre mail dans son intégralité si **au delà de 10 jours** je ne vous ai répondu, en précisant qu'il s'agit pour reprendre la formulation d'un ancien étudiant "**un petit UP**".

Il peut aussi m'arriver d'être pris par une recherche scientifique, voire par d'autres travaux et obligations universitaires, ou par quelques soucis de santé (il ne faut pas vieillir :-). Je vous préviendrai individuellement ou collectivement si je prends un peu de retard ou si je diffère la correction ou votre question. De manière générale, je privilégie en fin de semestre, **les étudiants les plus avancés** et surtout celles et ceux qui me soumettent **leurs projets terminaux**, ou les chapitres les plus ardu. Pendant, cette attente, rien ne vous empêche de continuer vos lectures et exercices, mais **attendez toujours** ma réponse pour m'adresser le chapitre suivant.

Remarque : Pour les étudiants en L2 et L3, vous avez reçu certainement des indications individualisées de la part des autres enseignants et intervenants. Comme indiqué précédemment, il vous suffira de m'envoyer un mail pour me notifier de votre commencement du cours. Bien évidemment, si vous vous étiez arrêté(e)s à un chapitre particulier, vous pouvez poursuivre l'année suivante avec celui-ci et/ou le dernier chapitre validé.

3 - Présentation Générale des Exercices

Pour chaque chapitre, vous aurez une série d'exercices à réaliser et une autre à me renvoyer. Elle est précisée, pour chacun d'eux, à la fin de celui-ci.

Adressez-moi vos exercices sous la forme d'**un unique document PDF**, non verrouillé, qui sera attaché à votre mail. Celui comprendra :

- **Une première page de présentation** comportant, dans cet ordre, vos prénom, nom et ID, le titre du chapitre s'il y en a un, accompagné de son numéro. Placez ces informations dans le quart supérieur. Vous laisserez le reste de la page vide, ce qui me permettra d'y insérer remarques et annotations ;
- Dans les pages suivantes, les intitulés complets des exercices et leur numéro ainsi que la question (par un simple copier-coller) ;
- **Vos réponses complètes et surtout commentées** (lire plus loin, les critères d'évaluation).
- Lorsque la réponse comporte du **code ou des fichiers annexes**, rassemblez tous les documents dans un ou plusieurs dossiers (de manière générale en reprenant le numéro des exercices comme nom) et que vous placerez **dans une archive unique** aux formats ZIP ou TGZ (évitez les formats trop exotiques) ;
- Donnez à votre document PDF, **le même nom que l'objet du mail** (IED). Faites de même avec le nom de l'archive si elle est présente ;

- Pour les projets finaux ou certains chapitres comportant beaucoup d'exercices avec développements et codes, engendrant des documents comportant un grand nombre de pages ajoutez **une table des matières paginée**.

Afin que vous ne perdiez pas trop de temps, adoptez toujours **la même feuille de style** dans votre traitement de texte pour tous les rendus d'exercices. Et pour les puristes et toutes celles et ceux qui envisagent de poursuivre vos études après la Licence, je vous invite vivement à vous familiariser puis à utiliser L^AT_EX pour vos rendus d'exercices.

Remarque : D'une manière générale, il est important de soigner la première de couverture de tout document puisqu'elle est constituée le premier contact avec le lecteur. Comme dans toutes syntaxe informatique, textuelle et ligne de code, essayez d'être intransigeant avec l'orthographe et la présentation.

4 - Réponses et Critères d'Evaluation

Comme je vous l'ai dit précédemment, je resterai dans la continuité des enseignants de l'IED qui, depuis le début, ont adopté **une notation globale pour l'ensemble de vos travaux**. La note globale tiendra compte :

- De votre investissement dans les différents chapitres. N'hésitez pas à dépasser les consignes des exercices, et à rendre les Bonus ;
- De la qualité de nos échanges, critère relationnel important ;
- Des petits plus que vous proposerez, c'est-à-dire la "valeur ajoutée". Voir quelquefois les petits défis "Bonus" que je vous lancerai ci-et-là pour éprouver votre perspicacité :-);
- Et Surtout de la**qualité des documents** que vous m'adressez : présentation, clarté, beauté et rigueur... en plus des résultats. Ajoutez le plus possible des **copies d'écran légendées** (c'est important), schémas, "extraits de pages"... N'oubliez pas de **citer toutes vos sources** (liens URL, etc.). Sachez que j'y suis sensible...

- A)** Apprendre, c'est aussi faire des erreurs. Nous en faisons tous. En général, je vous les indique par retour de mails et vous invite à me renvoyer une "**version 2**" (et donc d'ajouter "V2") à **votre PDF et à votre objet de mail**. Je tiens alors compte de cette V2 pour la notation et mesure les écarts (vos progrès) avec la "V1".

- B)** Apprendre, c'est mieux à plusieurs que tout seul. D'ailleurs, nous n'apprenons jamais seul à proprement parler. Les spécialistes des Sciences de l'Education donnent à ces interactions le nom de "Conflit Socio-Cognitif". Rien ne vous empêche, (et je vous y encourage), du moins dans les cours dont j'ai la charge, de vous contacter les uns, les autres et de vous entraider (entraider et non plagier :-). Vous pouvez aussi "interroger dynamiquement les nombreux forums dédiés (Stack Overflow, Site du Zéro...), prendre des informations dans les cours en ligne (Pluralsight, Udemy, Coursera, FUN, Lynda ou LinkedIn Video, Wintellect Now, EduCBA, Udacity, Video2Brain, MOOC d'autres universités,...

Je vous demande juste de le préciser explicitement sur votre (vos) exercice(s).

Par exemple : "**En m'appuyant sur le cours de X à l'URL suivant, j'ai trouvé ...**", "**j'ai posé la question à Y (à tel(le) étudiant(e)) ou à Z forum qui m'a donné cet élément de réponse suivant...**", "**l'exercice N.N a été fait en collaboration avec l'étudiant(e) prénom nom**" (**voir C**)....

Je préfère que vous preniez des initiatives de la sorte plutôt que d'écrire "*je ne sais pas répondre*", voire n'importe quoi ou une réponse hasardeuse. N'oubliez cependant de vérifier ET de valider l'information et/ou l'aide que vous avez trouvée !

- C)** Travailler seul.e voire isolé.e c'est particulièrement difficile et demande abnégation, persévérance pour ne pas dire obstination. Je ne suis pas opposé, si vous le souhaitez, que vous puissiez travailler un (ou un petit nombre ≤ 3) de chapitres **en binôme**, excepté le chapitre final (le projet) terminal qui sera traité individuellement. Dans ce cas, vos PDFs devront être à vos deux noms et envoyés avec vos deux adresses en adaptant : "*Chapitre N - Prénom1 Nom1 en coll. avec Prénom2 Nom2*". J'enverrai la correction et mes remarques en copie aux deux membres du binôme.

- La **correspondance** que j'entretiens avec vous est **individuelle** (hormis bien évidemment les mails collectifs), voire quelquefois relève de l'ordre de l'intime. Evitez, cela paraît trivial, de diffuser dans les différents réseaux (wiki, discord,...) les extraits de mails. Vous pouvez néanmoins partager les sources et les conseils méthodologiques sans oublier de citer leurs auteurs).

5 - Nouvelle Nomenclature des Exercices

Depuis la rentrée 2019-2020, une nouvelle nomenclature des exercices a été instaurée de manière à poursuivre la philosophie initiale de l'IED, à savoir développer des compétences initiales et un socle commun : En tenant compte de vos connaissances et talents à travers une progression individualisée, vous trouverez des exercices "Pivots" indiqués (**A Rendre**), des exercices (**Au Choix**), qui comme leur nom d'indique sont à réaliser au choix, et pour celles et ceux qui sont le plus à l'aise, des incitations à vous dépasser (**les Bonus**). Cette méthodologie de type "parcours" est reprise dans tous les cours dont j'ai la charge.

Précisez bien dans les intitulés des exercices de votre PDF les qualificatifs "**A Rendre**", "**Au Choix**" pour les exercices que vous avez choisis et "**Bonus**" pour le ou les exercices complémentaires..

Remarques : Gardez l'ordre original de la numération des exercices. Et c'est logique, il faut avoir réalisé l'ensemble des exercices prescrits au préalable avant de répondre au(x) exercice.s bonus.

Cette proposition de parcours individualisé me semble intéressante à plus d'un titre :

- Un ou deux exercices (**A Rendre**), parce que représentatifs et "pivots" pour chaque chapitre ;
- Des exercices au choix selon vos compétences, votre investissement temporel ou les difficultés pressenties, qui sont complémentaires aux exercices pivots ;
- Moins d'exercices à rendre, mais plus à expliciter et à soigner (ce qui a ma préférence) ;

- Des exercices Bonus "pour aller plus loin", pour les étudiant.e.s qui se sentent à l'aise et/ou qui aiment les défis et la note maximale (>15);
- Un temps plus conséquent attribué au(x) projet(s) qui représente(nt) plus de la moitié de l'évaluation terminale ;
- A terme, de limiter un peu plus les V2 (cf 5), les attentes de correction, et par conséquence, de diversifier un peu plus les corrections.

Pour cette nouvelle maquette, j'ai souhaité harmoniser le plus possible l'organisation des cours en dix parties ou chapitres de manière à raccourcir l'étalement du cours sur le semestre universitaire. En revanche, les chapitres sont un peu plus longs et plus complets que les années passées. En guise d'exemple illustratif pour une notation finale : les exercices de la progression des chapitres 1 à 9 (en vérité 10) validés vous attribue 10 points, le projet finalisé en apporte 5 points. Ce qui donne une note globale de 15 /20. Les points supplémentaires étant ceux gagnés par les bonus.

6 - Quelques Ressources Intéressantes

La bibliothèque de l'Université Paris 8 - Vincennes propose un grand nombre de ressources en ligne. Parmi celles-ci, SpringerLink, Wiley, et surtout **ENI Informatique** où vous avez l'intégralité des ouvrages de cet éditeur en accès intégral (en français)

-> Une collection d'anciens livres de l'éditeur **O'Reilly** est en accès libre (mais c'est souvent dans les vieilles marmites...) : [ici](#)

-> Le site **GDB** propose IDE, compilateurs et débogueurs en ligne pour les langages les plus courants (C, C++, Python, Java, PHP, Ruby, Perl, C#, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly GCC (cf L3), HTML, CSS, JS, SQLite, Prolog, Swift, Go, Rust, Bash...) :

-> La chaîne Youtube **Human Talks** propose des conférences sur des sujets informatiques actuels.

-> Les utilisateurs.trices de Python 3 peuvent s'ils (si elles) le souhaitent réaliser tous les exercices en utilisant **Jupyter Lab** (Ipython Notebook) ou encore **interact** ou encore **Google Colab** et me renvoyer leur(s) chapitre(s) aux formats **.ipynb et PDF**

->Le Site **GeeksForGeeks** vous offre également de nombreux outils et tutoriaux dans un grand nombre de langages.

7 - Présentation des Codes Sources

Même si cela n'apporte rien en termes de performances et d'exécution, pensez à harmoniser vos sources en y ajoutant une **entête** avec des **métadonnées** descriptives :

```
# ****
# Nom ..... : monProg
# Rôle ..... : .../... à écrire
# Auteur ..... : Mon Prénom, mon Nom
# Version ..... : V0.1 du 12/12/2020
# Licence ..... : réalisé dans le cadre du cours de ...
# (.../..)
# Compilation : gcc -Wall .../... (selon le langage utilisé)
# Usage : Pour exécuter : ./monProg param1 param2 param3
# *****/
```

Un grand nombre de réponses à des questions factuelles ou de codages que les programmeurs se posent sont dans le forum **Stack OverFlow**. N'hésitez pas à vous y rendre et à interroger son **moteur de recherche**

Bien que la plupart des exercices sont en C ou demande de coder en C, vous pouvez proposer vos réponses dans le langage 'open-source' de votre choix : Python3, Go, Rust, ... N'oubliez pas de préciser les modalités d'installation et d'utilisation (voire de compilation) et de joindre les sources en archive attachée.

8 - Quelques Remarques Récurrentes

Avant de conclure ce "Petit Guide", vous trouverez ci-dessous une petite **compilation des remarques** les plus courantes lors des années passées, pour vous aider à être plus efficient.e dans vos réponses :

R0) Prenez le temps de bien vous relire (et de vérifier vos calculs) avant

de m'envoyer votre PDF, car toute erreur triviale vous coûtera **de m'offrir un café :-)** le jour de votre soutenance.

- R1)** Les binaires s'écrivent toujours par **paquets de quatre chiffres** afin de favoriser la lecture et la conversion hexadécimale. (par ex : 1010 1101 1101 0011).
- R2)** Les décimaux quant à eux s'écrivent par paquets de **trois chiffres** : unités, dizaines et centaines. (par ex : 129 876 345).
- R3)** Lorsque vous citez dans vos réponses et que vous faites un copier-coller d'un extrait textuel dans une langue étrangère (souvent c'est en anglais), **ajoutez une traduction en français** de cette annotation ou de ce commentaire.
- R4)** Si vous devez fournir un code source, insérez dans le PDF celui-ci avec les commentaires¹ et faites un copier-coller du ou des sorties en prenant à minima **trois valeurs distinctes en y ajoutant idéalement une "valeur" supplémentaire qui générera une erreur en sortie** pour montrer l'étendu de son fonctionnement. N'oubliez pas de renseigner **les instructions de compilation**.
- R5)** Cela tombe sous le sens, mais pensez à toujours répondre à (la | toutes les) question(s) posée(s) dans l'énoncé à l'aide d'une ou de **plusieurs phrase(s) simple(s)**. Lorsque l'exercice comporte plusieurs questions ou que le développement de celui-ci est particulièrement long, ajoutez une conclusion en reprenant les différentes réponses sous la forme d'une synthèse finale.
- R6)** Certains exercices vous demandent des définitions, des RFC, de justifier votre réponse de manière "institutionnelle" ...**Citez toujours vos sources littéralement** (chemin complet) dans le corps de la réponse.
- R7)** Pensez à réaliser des **copies d'écran (C.E) les plus grandes et les plus lisibles possibles** et ajoutez toujours une légende.
- R8)** Parmi tous ces exercices, vous trouverez quelques questions véritablement subtiles, voire taquines (intellectuellement parlant :-)) que Jean Méhat avaient imaginées. Un grand nombre d'étudiants se sont heurtés à celles-ci sans pouvoir répondre "complètement" ou ont erré

1. Comme le soulignait Esther Schindler en 2009 : "If the comments are ugly, the code is ugly" :-) Lire à ce propos : <https://framablog.org/2009/11/21/commentaires-code-source-programmation/>). Il est vrai que "trop de commentaires" tue "le commentaire"... Mais comme nous sommes dans un rapport pédagogique et andragogique, ces commentaires même s'ils sont triviaux, me permettent plus facilement de suivre votre raisonnement...

dans moultes méandres et impasses. Ce sera une immense satisfaction pour vous si vous trouvez la réponse. En général, elle est comme les cils "*proches des yeux et pourtant on ne la voit pas*". Si vous y répondez et déjouez ses pièges :-), je ne manquerai pas de vous le faire remarquer et vous en féliciter !

- R9)** Copiez-collez (C.C) votre code source commenté dans son intégralité (avec les métadonnées descriptives) dans le corps de la réponse. S'il comporte un trop grand nombre de lignes pouvant gêner la fluidité de la lecture, placez les extraits "utiles" dans la réponse et la totalité de celui-ci reportée en Annexes.
- R10)** Une copie d'écran d'un Terminal (ou une figure) n'est qu'illustrative et ne peut être considérée comme une réponse en tant que telle. Reprenez (par copier-coller) l' (ou les) élément(s) de celle-ci comme les commandes saisies, les instructions de compilations, les paramètres passés, les sorties de programmes ou de commandes, ... textuellement dans une ou plusieurs petite(s) phrase(s).

Bonne lecture et de très bons TP(s) à vous...

Première partie

TP 1 - Outils collaboratifs - Git et (Concepts) Couverts

Chapitre 1

Git : l’Outil Collaboratif du Développeur

COMME le dirait Mandy Brown : "*Git est insupportable*". Mais il est également indispensable pour deux raisons aussi différentes qu'excellentes : il est très utile pour développer un site web ou une application à plusieurs, et au fil du temps, il est devenu une sorte de modèle pour la collaboration : des équipes distribuées qui travaillent de façon asynchrone dans un espace de travail commun.

Vous n'êtes bien évidemment pas obligés d'aimer Git, mais vous devez le connaître un petit peu. Outre les principales commandes et la syntaxe, un "commit" de modification n'est pas seulement un outil permettant de modifier du code, c'est aussi une pratique qui, judicieusement employée, permet de communiquer et de partager son travail. C'est un processus remarquablement puissant et il faut toujours avoir en mémoire que Git ne s'adresse pas à des machines, mais à des êtres humains...

Git est loin d'être la plus complexe des nouvelles technologies du Web, mais c'est une partie intégrante d'un bloc de base commun à tous les éléments de la pile. Il participe au nouveau Web des plates-formes et est devenu l'emblème du changement radical qui affecte le monde du développement.

Git est difficile en partie parce qu'il représente ce que Joel Spolsky appelle une "*abstraction percée*". Les abstractions, au sens logiciel du terme, consistent en une somme d'éléments qui rendent une tâche plus simple à manipuler conceptuellement, en dissimulant les éléments qui la rendent difficile. Les modèles scientifiques, les interfaces sont des abstractions : il



n'y a absolument aucun lien entre le geste qui consiste à faire glisser un fichier sur une icône de poubelle pour le supprimer et la suppression réelle du fichier sur votre disque dur. C'est simplement qu'un designer a pensé que cette métaphore rendrait le concept de suppression de fichiers plus compréhensible. Les abstractions sont là pour nous protéger de la complexité. Une *abstraction percée* ne remplit pas sa mission et laisse entrevoir la complexité sous-jacente, tout comme un parapluie percé ne remplit pas sa mission de vous tenir au sec...

L'interface de Git est "percée" car son interface en ligne de commande ne vous protège pas, en tant qu'utilisateur, de ce qui se passe sous le capot. Git possède sa propre logique, souvent assez différente de la façon dont nous, êtres humains, avons l'habitude d'organiser les informations. En d'autres termes, pour maîtriser Git, vous devrez penser comme Git...

Pour ce premier chapitre, la plupart des idées et l'approche des commandes sont tirées de l'ouvrage "GIT for Humans" de David Demaree (2016).

1.1 Qu'est-ce que la Gestion de Versions ?

Le principe de base de la gestion de versions consiste à garder une trace des différentes révisions au fur et à mesure, de manière à pouvoir, au besoin, revenir à une version antérieure, au lieu de conserver uniquement la dernière version d'un objet. La gestion de versions est avant tout une pratique et une méthode.

Il vous est certainement arrivé de conserver des versions antérieures de votre travail sur un document particulier, en utilisant l'option "Enregistrer sous..." de votre application pour donner un nouveau nom à chaque nouvel exemplaire, voire ajouté la date au nom de fichier. Ces deux approches sont des formes rudimentaires de la gestion de versions.

Les systèmes de gestion de versions comme Git fonctionnent en conservant un exemplaire de chaque version successive d'un projet dans un *repository* (ou dépôt), dans lequel nous pourrons valider (ou "*committer*"), des versions de notre travail qui représentent des pauses logiques, comme des points de sauvegarde dans un jeu vidéo. Chaque *commit* (ou validation de transaction) inclut des métadonnées utiles (nous avons à maintes reprises insisté sur l'ajout des méradonnées (cf Petit Guide et autres cours)) comme le nom et l'adresse e-mail de la personne qui en est l'auteur, ce



qui permet de savoir qui a réalisé telle ou telle modification particulière. Ces commits sont organisés en branches, dont chacune représente une piste d'évolution dans l'historique de notre projet, et parmi lesquelles une branche, appelée tronc ou branche maîtresse (*master*), représente la version officielle et principale. Lorsque nous bâtissons un historique de commits, il est ensuite facile de récupérer n'importe quelle version précédemment committée du projet, d'annuler des modifications ou de comparer deux versions ou plus à des fins de débogage.

Pour enregistrer des changements dans le repository, il faut une version du projet qui puisse être modifiée sans risque. Les systèmes de gestion de versions comme *Git* appellent souvent cela une *copie de travail*. Celle-ci joue le rôle de "bloc-notes" pour toutes les modifications que nous voulons apporter au projet. Nous pourrons ensuite "committer" ces changements dans le repository à titre de version officielle enregistrée. Une copie de travail est généralement facile à repérer : c'est **l'exemplaire qui se trouve sur notre disque dur**, sous la forme de fichiers conventionnels.

La gestion de versions peut sembler laborieuse parce que dans un processus de travail ordinaire, il faut enregistrer les modifications **à deux reprises** : à la fois *dans la copie de travail*, et *dans le repository*. La gestion de versions nécessite une pratique régulière pour porter ses fruits et devenir ce que David Allen (l'auteur de *Getting Things Done*), appelle un "*système de confiance*". D'un côté, une fois que nous avons committé une version de notre travail dans un repository, nous devons avoir la certitude que, la prochaine fois que nous la consulterons, elle sera exactement dans l'état où elle était précédemment. Et pour cela, nous devons prendre l'habitude de committer nos modifications à intervalles réguliers tout au long du développement.

Actuellement, le partage de fichiers tels que ceux de nos versions ne pose pas de difficultés. Le repository peut être un dossier partagé dans un service comme Dropbox ou Google Drive, ou encore un disque réseau ou un serveur local de fichiers. Pour inviter un nouveau collaborateur à notre projet, il suffit souvent de lui donner accès à ce dossier. Les choses se compliquent quand il faut non seulement synchroniser les modifications des différents membres de l'équipe, mais également coordonner ces changements de manière à ce que le système de gestion de versions reste fiable et viable. N'oublions pas que Git a été inventé par Linus Torvalds précisément pour répondre aux besoins exigeants du projet Linux, après un conflit de licence sur l'outil commercial de gestion de versions utilisé au-



paravant (voir article Wikipédia ci-dessous). Ces atouts font de Git l'outil idéal pour des projets tels que le noyau du système d'exploitation Linux qui compte des milliers de contributeurs et des centaines de milliers de commits. Mais Git sait aussi s'adapter avec grâce aux projets et aux équipes de petite envergure.

La Gestion de Versions sur Wikipédia et Atlassian



1.2 Les Commandes de Base

La ligne de commande, la même quelle que soit la plate-forme utilisée, est la langue maternelle de *Git*. Saisir des commandes et lire les réponses renvoyées par *Git* est, en outre, un excellent moyen d'apprendre le fonctionnement de celui-ci. Elle nous donne un accès total à l'intégralité des capacités de cet outil collaboratif. Notons qu'il existe un grand nombre d'applications proposant une interface graphique à *Git*, que nous ne développerons pas dans ce chapitre, mais que vous pouvez tester si vous le souhaitez (voir par exemple : *gitk*, *gitgui*, *GitHub*, *Git Force*, *Gitkraken*, *GitVine*, *Sourcetree*, *Tortoise Git*, *Git Cola*, ...). La plupart des IDE actuels comme *VS Studio*, *Suite Jetbrains*, *Xcode*, *Atom*, *Eclipse*... intègre le contrôle de versions avec *Git*.

Installation de Git



Les commandes de *Git* seront toutes plus ou moins de la forme suivante :



Forme Générale

```
1 ( master ) git commande paramètre1 paramètre2 ... —option
```

Si certaines fonctions de *Git* se suffisent d'un nom de commande (comme **git status**), la plupart nécessitent des paramètres pour faire leur travail. Bon nombre de commandes Git se lisent comme une sorte de phrase : "Git, fais ceci à cela". Par exemple, la commande **git checkout master** (vous trouverez actuellement "main") signifie essentiellement : "Git, extrais la branche nommée master" (ou main selon).

Les options sont des paramètres spéciaux signalés par au moins un tiret de tête. Elles sont rarement nécessaires et changent souvent la manière dont Git gère une tâche particulière. Beaucoup d'options ont une forme longue, comme **-global** et une forme raccourcie, comme **-g**. Certaines options prennent même une valeur, comme par exemple **git commit -message="Un Grand Bonjour de l'IED"**.

Git possède des centaines d'options de configuration, mais seules deux sont absolument nécessaires à son fonctionnement : le nom et l'adresse e-mail. Git ajoute à chaque commit un attribut Author comprenant ce nom et cette adresse e-mail, afin que les collaborateurs d'un projet puissent savoir qui a apporté une modification donnée. Le nom que nous saisissons servira à l'identification dans les logs (ou journaux de modification) et partout où Git affichera l'auteur d'un changement, tandis que l'adresse e-mail permettra non seulement à d'autres participants de vous joindre, mais indiquera également notre identité à un service hébergé comme GitHub.

La commande **git config**, contrairement à de nombreuses commandes Git, qui ne fonctionnent qu'au sein d'un projet Git, peut être exécutée depuis n'importe quel répertoire :

la Commande git config

```
1 git config —global user.name "Prénom Nom"  
2 git config —global user.email "Prenom.Nom@monfai.com"
```

L'option **-global** indique à *Git* qu'il s'agit de valeurs par défaut pour tous



les projets que nous développerons sur notre machine. Une brève remarque au sujet de la confidentialité car le partage d'informations personnelles est un sujet sensible : *Git* utilise le nom et l'adresse e-mail que nous fournissons pour attribuer le crédit des commits que nous effectuons. Dans le cas d'un repository local, ou tant que les commits n'ont pas été "poussés" sur un serveur, ces informations résident uniquement sur notre ordinateur. Mais les commits sont faits pour être partagés : nous sommes libres de fournir uniquement les informations personnelles que nous souhaitons, en utilisant un pseudonyme et une adresse pourrielle au besoin.

1.3 Crée un Nouveau Projet

La création d'une nouvelle *base de données Git* se réalise à l'aide de la commande **git init**. Pour suivre les modifications au sein d'un dossier de projet, il faut d'abord avoir un dossier de projet. Il peut s'agir d'un dossier existant dans lequel nous travaillons déjà et dans lequel nous ajoutons *Git*, ou bien d'un tout nouveau projet pour lequel nous allons utiliser *Git* dès le départ.

la Commande git init

```
1 mkdir mon_site_web
2 cd mon_site_web
3 git init
4 Initialized empty Git repository in /Users/demo/
   mon_site_web/.git/
5 (master) :
```

les Paramètres et Options de git init





1.4 Cloner un Projet Existant

Pour récupérer une copie du repository entreposé sur un serveur quelconque, nous allons utiliser une seule et même commande : **git clone**. Comparable à une macro, git clone est une commande pratique qui exécute plusieurs commandes connexes simultanément : elle crée un nouveau répertoire de travail (portant le nom du repository présent sur le serveur par défaut), initialise un nouveau *repository Git*, ajoute un serveur distant (remote) appelé **origin** et extrait les modifications de ce serveur distant.

la Commande git clone

```

1 mkdir mon_site_web
2 git clone https://github.com/github/copilot-docs
   mon_site_web.git
3 Cloning into mon_site_web...
4 remote: Counting objects: 11, done.
5 remote: Compressing objects: 100% (7/7), done.
6 remote: Total 11 (delta 1), reused 11 (delta 1)
7 Unpacking objects: 100% (11/11), done.
8 Checking connectivity... done
9 cd mon_site_web
10 (master) :

```

les Paramètres et Options de git clone



1.5 les Commits

Git s'attend à ce que chaque modification effectuée au sein d'un répertoire sous son contrôle soit enregistrée et stockée dans le repository. Sur le plan pratique, Git s'inquiète plus de la gestion de nos commits que des fichiers que nous modifions à l'aide de ces commits : son modèle d'organisation et de gestion de notre travail est axé sur les commits.



Les commits sont un type de données Git appelé objet. À l'intérieur, chaque information connue par Git : le contenu des fichiers, la structure des dossiers et, surtout, les commits qui signalent les autres comme versions d'un projet, tout cela est stockée sous la forme d'un objet, et chaque objet possède un nom unique dérivé de son contenu. En réalité, le nom d'un objet est plutôt un identifiant court, lisible par la machine, qui le distingue de manière fiable des autres objets. Chaque *commit* représente un snapshot (ou instantané) complet de l'état de notre projet à un moment précis ; son identifiant unique permet de distinguer cet état de l'aspect qu'avaient les fichiers de notre projet à tout autre moment.

Git procède par addition. Bien que des fichiers puissent être créés, supprimés ou modifiés, les commits qui consignent tous ces changements sont toujours ajoutés. Quand nous supprimons un fichier, nous ajoutons un commit. Si nous modifions une ligne de texte ou de code, ou que nous renommons un fichier, nous modifions l'état du projet et nous ajouterons un *commit* qui signalera ce changement et le propage au reste de votre équipe.

Les éléments de la base de données de *Git* sont inaltérables, à savoir qu'ils ne peuvent jamais être vraiment modifiés, seulement augmentés. *Git* est un système d'accumulation. En effet, *Git* additionne chaque changement que nous lui signalons pour que nous puissions revenir en arrière et explorer l'historique au besoin.

Nous venons de créer, dans la section précédente, un répertoire pour accueillir notre projet de site web. Nous allons maintenant ajouter un fichier qui servira de page d'accueil :

Une Page HTML très Simple

```
<!DOCTYPE html>
<html>
  <head>
    <title>Notre Super Site Web Collaboratif</title>
  </head>
  <body>
    <h1>Notre Site Collaboratif</h1>
  </body>
</html>
```



Ce fichier sera ensuite enregistré dans notre répertoire de projet sous le nom : **index.html**. Avant de continuer, nous pouvons demander à Git ce qu'il sait de l'état de notre projet en utilisant la commande **git status** :

la Commande git status

```
1 (master *) : git status
2 # On branch master
3 #
4 # Initial commit
5 #
6 # Untracked files:
7 # (use "git add <file>..." to include in what will be
8 # committed)
9 # index.html
10 nothing added to commit but untracked files present
11 (use "git add" to track)
```

Nous voyons ici plusieurs informations très intéressantes. Tout d'abord, nous nous trouvons dans la branche *master* du projet. Comme c'est un nouveau repository vide, notre prochain commit sera considéré comme le commit initial, le tout premier de notre chronologie. Enfin, et surtout, nous avons une liste de fichiers *untracked* (ou "non suivis"), dont notre fichier **index.html**.

Avant de pouvoir committer un fichier, celui-ci doit être suivi. Et pour suivre un fichier, il faut l'ajouter à la base de données de Git. Un commit enregistre les modifications apportées aux fichiers figurant dans la base de données de Git ; il indique par exemple qu'un fichier est passé de la version A à la version B (ou, dans le cas d'un commit initial, il définit ce qu'est la version A). En toute logique, pour savoir comment la version A a changé pour devenir B, Git doit connaître les versions A et B...

Comme le suggérait la commande **git status**, nous allons utiliser la commande **git add**. Ainsi, pour ajouter notre fichier HTML :

**la Commande git add**

```
1 (master *) : git add index.html
```

En général, Git ne répond rien. Mais comme c'est notre premier commit, nous allons vérifier que tout s'est bien déroulé avec **git status** :

la Commande git status (pour vérifier le "add")

```
1 (master *) : git status
2 # On branch master
3 #
4 # Initial commit
5 #
6 # Changes to be committed:
7 # (use "git rm --cached <file>..." to unstage)
8 #
9 # new file:   index.html
10 #
```

Notre fichier ne figure plus dans la liste des *untracked files*, mais dans la liste "*Changes to be committed*". Nous venons d'indexer notre fichier (avec add) et nous sommes prêts à le committer avec **git commit** :

la Commande git commit

```
1 (master *) : git commit --message "Commit initial"
2 [master (root-commit) 456FFC0 Commit initial
3 1 file changed, 9 insertions(+)
4 create mode 100712 index.html
5 (master) :
```

Sur la première ligne de la réponse, nous voyons l'identifiant unique de ce commit : (ici) 456FFC0. L'argument *—message* peut également être saisi sous la forme abbrégée *-m*. (par ex : *git commit -m "Commit initial"*)



les Paramètres et Options de git commit



Les trois commandes *status*, *add* et *commit* composent l'essentiel de notre interaction avec *Git*.

Nous allons réaliser un second commit illustratif en ajoutant un fichier CSS très basique qui sera utilisé par notre page d'accueil. Dans le répertoire de notre projet, nous allons créer un sous-répertoire appelé "css" avec **mkdir** et le style ci-dessous sera sauvegardé dans le fichier **style.css**.

Une Feuille de Style très Simple

```
body {  
    font-family: 'source-code-pro', Arial, sans-serif;  
    font-size: 100%;  
}
```

Voici une "nouvelle version" notre fichier index.html :

Une Page HTML très Simple et Modifiée

```
<!DOCTYPE html>  
<html>  
    <head>  
        <title>Bonjour</title>  
        <link href="css/styles.css" type="text/css" media="all">  
    </head>  
    <body>  
        <h1>Bonjour à Toutes et à Tous, les Etudiants de l'IED</h1>  
    </body>
```



```
</html>
```

Nous venons d'ajouter un répertoire (css/) contenant un fichier (styles.css), puis nous avons modifié un fichier existant (index.html) . Vérifions maintenant avec git status :

la Commande git status (pour vérifier les changements)

```

1 (master *) : git status
2 # On branch master
3 # Changes not staged for commit:
4 # (use "git add <file>..." to update what will be
5 # committed)
6 # (use "git checkout -- <file>..." to discard
7 # changes in working directory)
8 #
9 # modified:   index.html
10 #
11 # Untracked files:
12 # (use "git add <file>..." to include in what will be
13 # committed)
14 #
15 # css/
15 no changes added to commit (use "git add" and/or "git
commit -a")
```

Sous l'intitulé "*Changes not staged for commit*", nous retrouvons le fichier HTML que Git connaît déjà et qu'il présente maintenant comme modifié. En dessous se trouve à nouveau la liste "*Untracked files*" mais à la place du fichier styles.css ajouté précédemment, figure le répertoire *css/* qui le contient. Git indique de cette manière qu'il ne connaît pas du tout ce sous-répertoire.

Ces deux changements peuvent être indexés à l'aide d'un simple **git add**. Nous allons maintenant indiquer à la fois le fichier HTML et tout le répertoire *css/* comme arguments en les séparant par une espace pour indiquer qu'il faut indexer séparément chaque élément de la liste :



les Commandes git add et status

```

1 (master *) : git add index.html css/
2
3 # puisque Git est muet, nous verifions avec git status :
4
5 (master *) : git status
6 # On branch master
7 # Changes to be committed:
8 # (use "git reset HEAD <file>..." to unstage)
9 #
10 # new file:    css/styles.css
11 # modified:   index.html
12 #

```

Les deux fichiers sont indexés et prêts à être committés. Git a remarqué que *index.html* a été modifié par rapport à une version précédente, tandis que *styles.css* (et le répertoire *css/* où il se trouve) sont de nouveaux éléments dans ce commit :

Un Second Commit

```

1 (master *) : git commit -m "Ajouté feuille de styles"

```

Grâce à ce second commit, notre projet a maintenant un historique. Notons qu'il est possible de combiner les options *-a* et *-m* en *-am* : *-a* (pour ajouter automatiquement tous les fichiers modifiés au commit) et *-m* (pour préciser un message de commit) par exemple (*git commit -am "Modifié couleur de fond de ma-page-web"*)

1.6 Quelques Commandes Supplémentaires

- (**git rm**) Il est important de comprendre que si nous supprimons un fichier de la copie de travail d'un projet, il faut également le retirer du repository. La commande *git rm*, s'applique à faire croire qu'elle se contente de supprimer un fichier. Cependant Git est un système à accumulation, et Git ne se soucie des changements que dans le contexte d'un commit. Quand nous supprimons un fichier avec *git rm*, Git crée un nouveau snapshot du projet sans le fichier supprimé et indexe cette version comme la prochaine



à committer. Après une suppression, n'oublions pas de committer : (git commit -m "Retiré fichier style.css").

- (**git add -all**) (ou git add -A en abrégé). L'option -all est idéale pour committer plusieurs changements simultanément.

- (**git branch**). Chaque repository Git commence avec une branche principale à laquelle *Git* donne le nom de *master* (ou *main* par défaut. Sur le plan technique, *master* (ou *main*) est une branche comme les autres ; ce qui la rend spéciale est seulement son rôle conventionnel de version primaire, stable, du projet qui se trouve dans un repository. Pour afficher la liste de toutes les branches du repository local, nous utiliserons la commande *git branch*. Pour créer une nouvelle branche, il suffit de passer son nom dans la commande *git branch* : (git branch ma-nouvelle-page-accueil).

► Attention, *Git* ne bascule pas automatiquement dans la nouvelle branche à l'exécution de la commande (git branch ma-nouvelle-page-accueil). *Git* crée une nouvelle branche mais *master* (ou *main*) reste la branche en cours ou (en termes *Git*) la branche extraite. Si nous restons ainsi, *Git* a créé la nouvelle branche, mais le prochain commit sera fait dans la branche *master* /*main*...

Si vous êtes débutant.e. dans la technologie et dans les concepts de branches manipulées par *Git*, je vous invite à vous reporter aux les deux liens ci-dessous (SCM et Atlassian).

les Branches (à lire) et Options de git branch



- (**git checkout**). Pour changer de branche, il faut utiliser la commande *git checkout*. (git checkout ma-nouvelle-page-accueil). Créer une branche puis y basculer ensuite peut devenir fatigant. Heureusement, *Git* propose un raccourci pratique : la commande *git checkout* pour créer une nouvelle branche et y basculer en une seule opération, grâce à l'*option -b* : (git checkout -b ma-nouvelle-page-accueil)



les Paramètres et Options de git checkout



- (**git merge**). Il arrive parfois qu'une branche soit simplement le lieu d'expérimentations. C'est l'un des grands avantages de l'embranchement dans les systèmes de gestion de versions en général et dans Git en particulier : la création d'une branche est rapide et peu coûteuse, et nous ne sommes jamais tenu de réconcilier la version de travail d'une branche avec celle de master (ou main). La fusion (ou merge) combine deux (ou plus, mais généralement deux) branches différentes de notre projet en une version unifiée présentant les attributs des deux branches : (git merge version-beta-main-page).

► Le type de merge le plus simple et le plus facile est appelé **fast-forward**, et correspond en effet exactement à une "avance rapide". C'est celui que vous rencontrerez le plus souvent.

les Paramètres et Options de git merge (à lire)



1.7 Collaborer : Les Repositories Distants

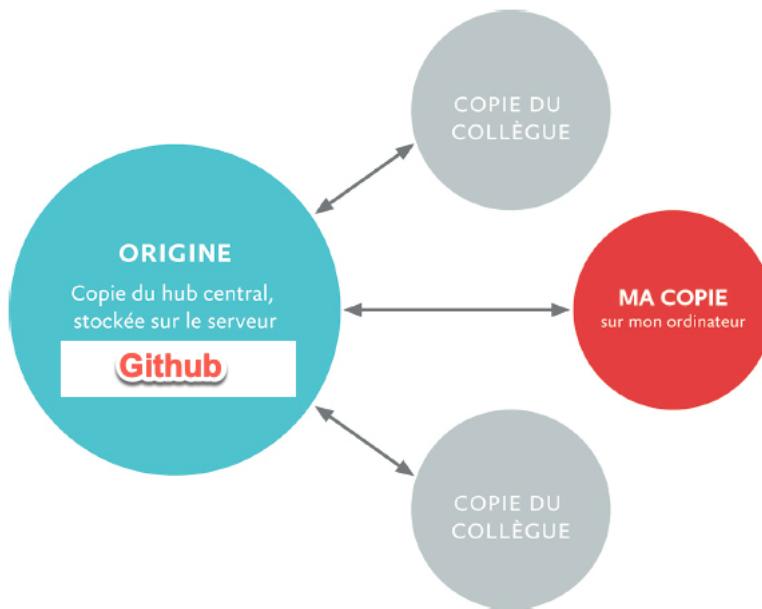
Les modifications nous avons effectuées jusqu'alors résidaient dans un seul et même endroit : notre ordinateur. Pour les projets développés en local ou individuellement, nous pouvons ainsi bénéficier de toute la puissance de la gestion de versions sans avoir à accéder à un serveur ou à créer un compte quelque part. Mais ce n'est pas pour travailler seul que nous faisons appel à Git, c'est pour collaborer.

Un repository distant, par opposition à un repository local sur notre ordi-



nateur, est une copie d'un projet Git qui se trouve ailleurs : soit sur une autre machine de notre réseau, soit sur l'ordinateur d'un collaborateur, sur un service en ligne comme GitHub. Les repositories distants représentent l'une des abstractions les plus puissantes de Git. Contrairement aux branches, qui sont pour l'essentiel des copies virtuelles de notre projet, chaque repository distant correspond à une copie réelle et physique de notre repository, avec laquelle nous pouvons échanger des données. La plupart des opérations que nous aurons besoin de réaliser pour échanger des modifications avec un repository distant tiennent en deux commandes : **push** (pousser) et **pull** (tirer), qui produiront l'effet escompté.

La conception décentralisée de *Git* vous permet de pousser et de tirer des modifications entre deux ordinateurs, quels qu'ils soient : par exemple, nous pouvons directement pousser les commits d'une branche résidant sur notre ordinateur vers une branche de l'ordinateur d'un collègue ou d'un.e autre étudiant.e de l'IED, et inversement. Il est cependant plus aisés de partager son code via Git, à l'aide du "modèle de hub". Ainsi, nous et notre équipe conserveront une copie commune du projet sur un serveur distant (le hub), accessible à tous les membres de l'équipe. Chaque collègue qui se joint au projet copie (ou clone) le repository du projet sur son ordinateur, effectue des modifications et des commits, puis utilise les commandes **git push** et **git pull** pour synchroniser le repository avec celui du serveur.



- Selon le modèle de hub, les membres de l'équipe synchronisent leur copie locale avec une copie centrale commune plutôt que les uns avec les autres. -

Le Hub joue également le rôle de sauvegarde fiable du code au cas où la copie d'un collaborateur serait endommagée ou perdue, ou si un utilisateur souhaite récupérer une copie de son travail sur une nouvelle machine. Au lieu de copier les fichiers d'un ordinateur à un autre, il est souvent plus simple de cloner le projet Git sur la nouvelle machine à partir du Hub.

Pour bien comprendre l'utilisation de la commande **git remote** et de toutes ses *sous-commandes* comme **remote add**, **remote rm**, ... prenez le temps de lire les deux articles ci-dessous dont les liens sont en QR-code (SCM et Atlassian).

les Paramètres et Options de git remote





git remote

```

1 git remote add origin https://github.com/mon_nom/mon_repo.
    git
2 # Set a new remote
3
4 git remote -v
5 # Verify new remote
6 > origin https://github.com/mon_nom/mon_repo.git (fetch)
7 > origin https://github.com/mon_nom/mon_repo.git (push)

```

► Il est possible d'utiliser les protocoles SSH (par ex :)
git@github.com : mon_nom/mon_repo.git
et HTTPS (par ex :)
https://github.com/mon_nom/mon_repo.git
et de commuter l'un avec l'autre (par ex :)
`git remote set-url origin https://github.com/mon_nom/mon_repo.git`
ou encore
`git remote set-url origin git@github.com : mon_nom/mon_repos.git`).

Il nous suffit ensuite de saisir ces deux commandes en tapant sur la touche Entrée après chacune. Ces commandes permettent d'indiquer à Git qu'il doit gérer (de manière simplifiée ici) tous les fichiers du dossier :

Utilisation Facile de GitHub

```

1 # git remote add origin https://github.com/mon_nom/mon_repo
    .git (fait précédemment)
2
3 git add --all
4
5 git commit -m "ajout des fichiers dans notre dépôt"
6
7 # envoi sur GitHub :
8 git push -u origin master
9
10 # ou encore :
11 git push -u origin main

```

Et si vous avez un autre "commit" à réaliser :



Utilisation Facile de GitHub et des Commits

```
1 git add --all  
2 git commit -m "Un autre commit"  
3 git push
```

Github en quelques mots...

GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Il a été développé à l'aide de Ruby on Rails et du langage Erlang par Chris Wanstrath, PJ Hyett et Tom Preston-Werner. GitHub assure un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

Octocat est la mascotte de la marque. Il a été dessiné par Simon Oxley (il est également le créateur du logo de Twitter) en s'inspirant de l'art manga (des oreilles de chat et des tentacules de céphalopode).

En avril 2016, GitHub a annoncé avoir dépassé les quatorze millions d'utilisateurs et plus de 35 millions de dépôts de projets le plaçant comme le plus grand hébergeur de code source au monde. (voir Article Wikipédia dont ce résumé a repris les grandes lignes pour de plus amples informations).

Comme pour tout outil (ou tout langage), le plus simple est d'apprendre à l'utiliser à l'aide de tutoriaux. Avant de vous lancer dans la réalisation des deux exercices (A Rendre + Bonus) de ce TP, regardez les exemples et les astuces.

Tutoriels en Vidéo (YT) pour utiliser Git et GitHub

- Un tutoriel pour les débutant.e.s (La Capsule)



- Un tutoriel (sous la forme d'un TD) par (Le Wagon)



- Un dernier tutoriel (installation et utilisation pas à pas) par (From Scratch - Développement Web)



1.8 Les Exercices du TP

Pour l'ensemble des 4 TP(s) de ce "cours" (qui comme GIL n'en est pas vraiment un :-), juste quelques activités autour des outils collaboratifs, vous aurez besoin :

- de votre ordinateur personnel (accès au Terminal), avec les droits administrateurs, relié à Internet, sur lequel vous aurez à installer quelques logiciels (en version d'évaluation, en open source, ou gratuits), serveur web, et à utiliser un langage de programmation (sur certains exercices et Bonus) comme Python, JS, ou tout langage de votre choix. Ne pas oublier de préciser les options éventuelles de compilation ou d'exécution.

► Remarque : Il n'est pas nécessaire de disposer d'un ordinateur puissant dernier cri. Comme pour tous les cours de la Licence, un simple Raspberry Pi v3 ou v4 suffit.



Important

→ Les étudiant.e.s qui possèdent un compte Github et qui ont déjà utilisé Git pour déposer un ou plusieurs codes sources dans le cadre de l'un ou l'autre cours de la Licence Informatique **n'ont pas besoin de réaliser l'exercice 1.1**. Pour valider celui-ci, il vous suffira de préciser **l'URL de votre espace dans Gitbub, la thématique** de celui-ci et de fournir **une ou plusieurs Copie(s) d'Ecran (C.E(s))** de votre espace dans le corps de votre réponse.

Pour ce premier chapitre, vous devez ajouter dans votre PDF l'**exercice 1.1** (Lire l'encadré "Important"). L'**exercice 1.2**, qui touche au ludo-éducatif et au jeu collaboratif constructif est pour un Bonus bien mérité.

Exercice 1.1 (A Rendre)

A l'aide de Git et de Github :

☞ Réalisez si ce n'est déjà fait **une installation de Git** sur votre machine **en expliquant** les différentes étapes et effectuez **quelques commandes fondamentales** à l'identique du cours que vous commenterez.

☞ Déposez, après vous être enregistré.e, le(s) code(s)-source(s) de votre choix **sur GitHub** : par exemple, le code de votre serveur web (cf IF-Partie 2 (UOR), Codes issus d'ADO ou GIL, ou tout code (ou document textuel) qui aurait de l'intérêt pour vous. Proposez également **une ou plusieurs modifications** de celui (ou ceux)-ci pour montrer votre aisance dans **l'utilisation du versioning**.

✓ Les étudiants qui possèdent et utilisent régulièrement un espace Github, cet exercice est validé (voir encadré "Important")

✓ Vous pouvez, selon, utiliser le Terminal ou le client Git (ou encore une interface graphique), IDE,... de votre choix.

✓ Comme pour tous les exercices, pensez à bien **expliquer** votre méthodologie, vos choix. Citez vos **sources littéralement** dans le corps de votre réponse ainsi que les commandes utilisées et **illustrez** celles-ci à l'aide de Copies d'Ecran (cf Petit Guide).



- Fig.1 - Un monde Minecraft pour le Bonus -

Exercice 1.2 (Bonus)

Après avoir installé votre propre serveur **Minecraft** (voir les instructions d'installation ci-après) :

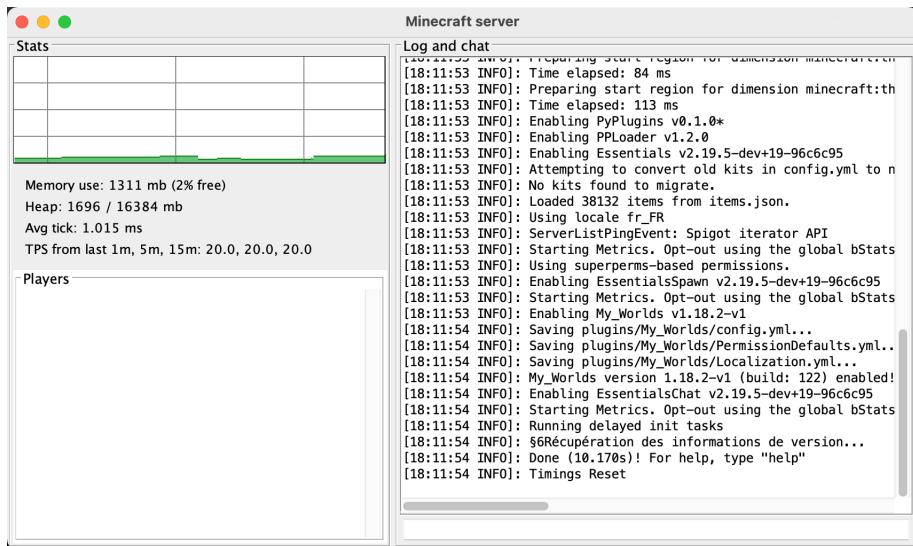
- » Avec la méthode de votre choix (Construction verticale, Maison, Red Stone, ...), **Faites apparaître** dans votre monde Minecraft **le nom de votre espace** (ou entrepôt) Github à votre joueur (aux joueurs connectés).
- » Insérez (ou faites apparaître) avec la méthode de votre choix, **l'adresse de votre espace Github** dont le lien sera **cliquable** par le ou les joueurs.
- » Pour les aficionado.aficionada uniquement, faites apparaître **la carte de votre monde** sur votre site web (en local) à l'aide du plugin **Dynmap** (ou équivalent). Vous pouvez utiliser par exemple celui que vous avez développé pour IF-Partie 2 (UOR) ou en créer un autre.
- ✓ Comme à l'habituée, pensez à bien expliquer votre méthodologie, vos choix et commandes utilisées, et d'insérer plusieurs C.E(s) illustratives.
- ✓ Pour la seconde question, une solution simple consiste à utiliser la commande "**tellraw**" comme suit :


```
/tellraw @a "text" :"Votre Message de bienvenue sur votre espace Github)","clickEvent" :"action" :"open_url","value" :"https ://adresse_de_mon_compte_github"
```
- ✓ Pour celles et ceux qui maîtrisent plus ou moins le **format JSON** (vous en aurez besoin pour les exercices du chapitre 2), vous



pouvez utiliser le générateur pour *tellraw* sur ce site : [Tellraw Generator for Minecraft](#)

✓ Pour la troisième question (qui vous apportera un **mini-Bonus supplémentaire**), voici un petit tutoriel : [Comment mettre en place Dynmap sur votre serveur Minecraft](#).



- Fig.2 - Démarrage du Serveur Minecraft -

Installer un Serveur Minecraft

- 0 - Le plus simple est de rechercher un [tutoriel en ligne](#) (site web, blog, wiki, YT,...) en français (ou dans la langue de votre choix) pour l'installation du serveur Minecraft. Deux ont retenu notre attention : [Spigot](#) et [Paper](#).

- 1 - Après avoir récupéré Paper (petite préférence) ou Spigot en version v1.18.(1 ou 2) par ex pour l'un et l'autre, vous aurez besoin d'installer, si ce n'est déjà fait le JDK JAVA 17 (ou OpenJDK) ou sup sur votre système. (Voir par exemple [ici](#))

- 2 - (Pour les puristes uniquement), le mieux est peut-être de générer l'exécutable java pour spigot (spigot-1.18.2.jar) à l'aide de l'outil "[BuildTools](#)" à partir des sources. (Un petit tutoriel [ici](#))

- 3 - Après avoir créer un dossier pour votre serveur, il est ais de le démarrer à l'aide de la commande dans le Terminal :



Lancement du serveur Minecraft

```
java -jar spigot-1.18.2.jar  
# ou java -jar paper-1.18.2.jar (selon)
```

- 4 - Comme l'indique la plupart des tutos (cf minecraftfacile...), la première compilation génère une erreur en raison de la non-acceptation de la licence associée. Avec l'éditeur de votre choix (Sublime Text, Notepad++, Vim, Emacs,...), éditez le fichier "eula.txt" et mettre la valeur "eula=true" (à la place de false), indiquant que vous avez pris connaissance de celle-ci.
- 5 - Faites de même en ouvrant le fichier de configuration "server.properties" avec votre éditeur de textes, et changez : "motd= Le Nom de Votre Serveur Minecraft" (par exemple ou laisser tel quel), le port (au besoin, inutile si vous n'avez pas d'autres serveurs utilisant ce port) et surtout "online-mode=false" (bien positionner à false pour une utilisation en local et/ou pour éviter de se connecter à un compte Mojang)
- 6 - Relancer le serveur (comme pour (3)) par : java -jar spigot-1.18.2.jar ou java -jar paper-1.18.2.jar (cf Fig.2)
- 7 - Une fenêtre de contrôle s'ouvre, affiche des informations sur le serveur puis dans la console, vous devriez voir une invite de commandes en forme de chevron dans le terminal : <. Saisissez "op (Username)" (+ entrée)(par exemple : op demo, si "demo" est le nom du joueur que vous avez saisi dans le TLauncher (voir ci-après (9))). Cette commande vous permet d'obtenir les droits "opérateur" de niveau 4 par défaut, soit les droits maximaux sur votre serveur. (Au besoin lire : [How to OP Yourself In Minecraft](#)) (cf Fig.3)
- 8 - Comme nous utilisons Minecraft uniquement dans un but pédagogique et démonstratif, vous pouvez télécharger le [TLauncher](#) et déposer l'exécutable JAVA (TLauncher-2.841.jar par ex à la version du cours) dans un nouveau répertoire de votre choix. Cet exécutable est disponible pour tous les systèmes d'exploitation (Linux, MacOS, MS Windows).
- 9 - Choisissez la version de Minecraft (client) que vous souhaitez utiliser (la même version que votre serveur démarré (pour nous c'est 1.18.2)), votre nom d'utilisateur dans "Compte" (le même qui possède les droits "opérateur", pour nous c'est "demo" par ex) et lancez

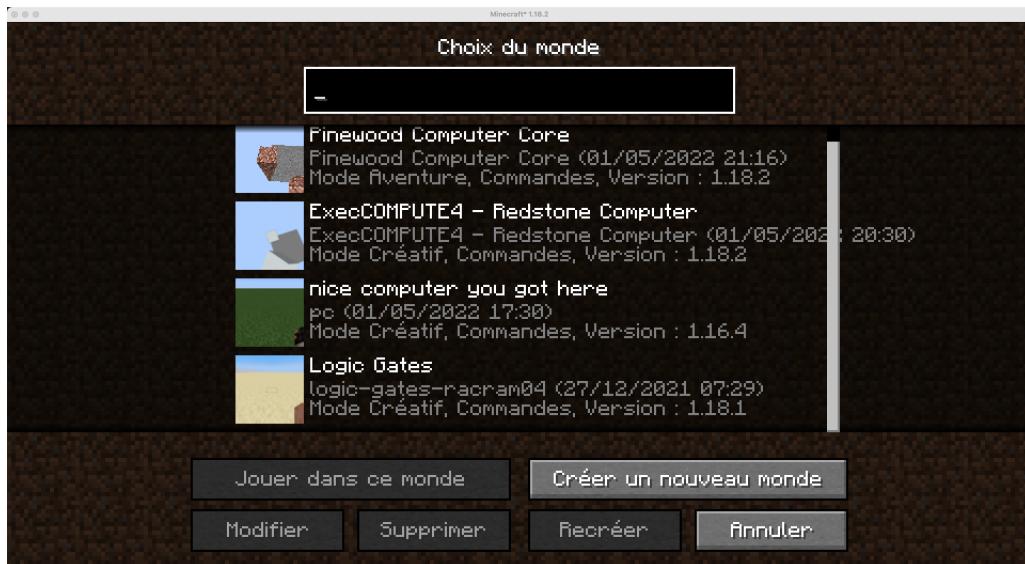


le programme (client) Minecraft. Au besoin, vous pouvez adapter les paramètres (taille de la fenêtre,...)

- 10 - Le lancement charge les fichiers du jeu (Minecraft version Java, ici la 1.18.2). Choisissez ensuite "Multijoueurs", puis "Nouveau serveur". Indiquez le nom de votre serveur (cf (5)) puis dans le champ adresse mettre l'IP locale de votre serveur (par ex : 192.168.n.n :25565). Ca y est, vous êtes connecté à votre serveur, et à votre monde. (fig.4)
- 11 - De manière optionnelle, vous pouvez installer des plug-ins de votre choix, comme par exemple **EssentialX**, des MODS comme **World-Edit**, ..., ou tout autre qui vous permettrait de répondre aux questions du Bonus.

```
[18:11:53 INFO]: [Essentials] Loaded 38132 items from items.json.
[18:11:53 INFO]: [Essentials] Using locale fr_FR
[18:11:53 INFO]: [Essentials] ServerListPingEvent: Spigot iterator API
[18:11:53 INFO]: [Essentials] Starting Metrics. Opt-out using the global bStats config.
[18:11:53 INFO]: [Essentials] Using superperms-based permissions.
[18:11:53 INFO]: [EssentialsSpawn] Enabling EssentialsSpawn v2.19.5-dev+19-96c6c95
[18:11:53 INFO]: [EssentialsSpawn] Starting Metrics. Opt-out using the global bStats config.
[18:11:53 INFO]: [My_Worlds] Enabling My_Worlds v1.18.2-v1
[18:11:54 INFO]: [My_Worlds] Saving plugins/My_Worlds/config.yml...
[18:11:54 INFO]: [My_Worlds] Saving plugins/My_Worlds/PermissionDefaults.yml...
[18:11:54 INFO]: [My_Worlds] Saving plugins/My_Worlds/Localization.yml...
[18:11:54 INFO]: [My_Worlds] My_Worlds version 1.18.2-v1 (build: 122) enabled! (0.691s)
[18:11:54 INFO]: [EssentialsChat] Enabling EssentialsChat v2.19.5-dev+19-96c6c95
[18:11:54 INFO]: [EssentialsChat] Starting Metrics. Opt-out using the global bStats config.
[18:11:54 INFO]: Running delayed init tasks
[18:11:54 INFO]: [Essentials] §6Récupération des informations de version...
[18:11:54 INFO]: Done (10.170s)! For help, type "help"
[18:11:54 INFO]: Timings Reset
> op demo
```

- Fig.3 - Démarrage du Serveur Minecraft -



- Fig.4 - Lancement d'un Monde (ici, il y en a plusieurs qui sont installés) -

Quelques liens utiles pour Minecraft



1.9 Quelques Liens pour Aller plus Loin...

Quelques Cours et Ouvrages Complémentaires à Explorer...

!

- (1) Dans la collection ENI, disponible en ligne sur le site de la B.U de Paris 8, l'ouvrage "Maîtrisez la gestion de vos versions (concepts, utilisation et cas pratiques) (3e édition)" de Samuel Dauzon (2021)





- (2) OpenClassRooms propose un cours intitulé "Gérez du code avec Git et GitHub" assez complet :



- (3) De même, Coursera propose une formation (en anglais) "Introduction to Git and GitHub" certifiante. Vérifiez en suivant le QR-Code ci-dessous, la date d'inscription :



- (4) Plusieurs formations et tutoriaux sur FUN (Gitlab, Jupyter, Rstudio (R), Python, ...)



- (5) Enfin, en complément des tutos vidéos cités plus haut, la chaîne Grafikart propose une formation complète sur Git en 18 épisodes.



Deuxième partie

TP 2 - Requêter à l'aide des API

Chapitre 2

API - Collecter de l'Information de Manière Collaborative

Ce deuxième TP aura pour objectif de collecter de l'information à l'aide des interfaces de programmation d'application appelées couramment API. Nous allons apprendre à utiliser plusieurs API(s), et principalement celle de Github et celle du site Hacker News...

2.1 Qu'est-ce qu'une API ?

Une **API** (Application Program Interface) est une interface normalisée et sécurisée qui permet aux programmes et aux applications de communiquer et d'échanger des données ou des informations collaborativement les unes avec les autres. L'API est spécialement conçu pour la récupération et la mise à jour "automatique" d'informations, c'est-à-dire sans intervention manuelle de l'utilisateur.

Les API s'organisent en deux architectures principales : **SOAP** (Simple Object Access Protocol) et **REST** (REpresentational State Transfer). Dans le cadre de ces notes de cours, **nous ne développerons pas SOAP**, plus ancienne et construite autour du **format pivot XML**. Actuellement, l'architecture d'API la plus couramment utilisée est **REST**, créée par Roy Fielding, et qui définit comment les applications peuvent communiquer via **HTTP(s)** pour transférer des informations de manière efficace et rapide. Avec les API dites **RESTful**, les communications entre programme(s) et API(s) sont faiblement couplées, ce qui signifie que chaque API (ou programme) n'a pas connaissance des définitions et des formats de données. Pour les connaître, il suffit bien entendu de consulter la documentation en



ligne...

Les API REST(ful) comportent cinq méthodes (ou actions) HTTP courantes (cf RFC 2616) :

- **PUT** : méthode utilisée pour modifier ou mettre à jour des données sur un serveur existant ;
- **POST** : Utilisée pour envoyer (ou créer) des données du client au serveur ;
- **GET** : Utilisée pour récupérer / collecter des données du serveur ;
- **PATCH** : Utilisée pour mettre à jour des données ou informations existantes sur le serveur ;
- **DELETE** : Utilisée pour supprimer des données, des informations sur le serveur.

De nombreux sites font appel à des API REST(ful), comme par exemple Google, Amazon, Twitter et LinkedIn pour ne citer que les principaux.

2.2 S'Entraîner aux Actions avec une API

Aujourd’hui, la plupart des plateformes informatiques en ligne sont accessibles à travers le protocole HTTP(s) et leurs données avec JSON (ou encore avec des formats comme XML ou CSV). Ces interactions en mode textuel sont aisées à comprendre et à programmer. Il est généralement dit que les API sont "**language agnostic**" car elles peuvent être accédées par n’importe quel langage informatique qui possède un module client HTTP, en incluant bien évidemment Python que nous allons utiliser ici.

La définition stricte de la propriété "RESTful" comporte les caractéristiques vues plus haut, mais d’une manière plus pragmatique, elle permet d'accéder aux ressources à travers une collection d'adresses (URL) appelée aussi "endpoint". Ainsi chaque adresse représente une ressource particulière comme un article, un journal, un livre, une image, une carte, un post ou un commentaire, etc. Ces ressources ou éléments informationnels sont "manipulables" à travers ces méthodes (GET pour voir, POST pour créer, PUT/PATCH pour éditer, et DELETE pour supprimer).

En guise d’entraînement et de découverte de ces actions (les mêmes globalement que pour un SGBD), nous allons utiliser la plateforme **JsonPlaceholder** qui simule ces interactions.



- Fig.1 - Free Fake API for Testing -

JSONPlaceholder nous propose de découvrir six resources :

Les Informations Disponibles

```
/posts 100 posts
/comments 500 comments
/albums 100 albums
/photos 5000 photos
/todos 200 todos
/users 10 users
```

Ainsi, en appliquant la logique des URL(s) :

Les Informations Disponibles

```
# La collection comportant toute les albums :
/albums
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



```
# Une photographie particulière . X étant l'ID(entifier) de cette photo :  
/photos/X  
  
# Les photos d'un album  
/albums/X/photos  
  
# ...
```

Le plus simple est de consulter le [guide](#) de cet API.

Exercice 2.1 (Pour Réfléchir)

Dans votre terminal, saisissez "python" ou "python3" (selon) de manière à afficher les trois chevrons du REPL "<<" :

☞ **Récupérez** tous les albums disponibles sur JSONPlaceholder avec GET. Faites de même avec les photos.

A saisir dans le REPL Python

```
1 # Python 3.8.3 (default, Jul 2 2020, 11:26:31)  
2 # [Clang 10.0.0 ] :: Anaconda, Inc. on darwin  
3 # Type "help", "copyright", "credits" or "license" for more information.  
4 <<< import requests  
5  
6 # si vous obtenez une erreur, installez requests avec pip (ou pip3)  
    install requests  
7  
8 <<< resultat = requests.get('https://jsonplaceholder.typicode.com/  
    albums')  
9 <<< resultat  
10<Response [200]> # la requête s'est bien déroulée  
11<<< resultat.json() # affichage des 100 albums au format JSON  
12[{'userId': 1, 'id': 1, 'title': 'quidem molestiae enim'}, {'userId': 1,  
    'id': 2, 'title': 'sunt qui excepturi placeat culpa'}, {'userId':  
    1, 'id': 3, 'title': 'omnis laborum odio'}, {'userId': 1, 'id': 4, '  
    title': 'non esse culpa molestiae omnis sed optio' ...}....
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



☞ Ajoutez un nouveau commentaire sur JSONPlaceHolder avec POST. Faites de même avec une nouvelle photo.

A saisir dans le REPL Python (suite)

```
1 <<< commentaire = {'userId': 10, 'title ': 'le titre de votre choix', 'body': 'le commentaire de votre choix à développer'}
2 <<< resultat = requests.post('https://jsonplaceholder.typicode.com/posts',
3                               json=commentaire)
4 <<< resultat
5 <Response [201]> # la requête s'est bien déroulée (code 201 -> Création)
6 <<< resultat.json() # affichage du POST
7 { 'userId': 10, 'title ': 'le titre de votre choix', 'body': 'le commentaire de votre choix à développer', 'id': 101}
```

☞ Modifiez un commentaire sur JSONPlaceHolder avec PATCH (ou le titre d'une photo, etc.)

A saisir dans le REPL Python (suite)

```
1 # Nous récupérons ici le dixième commentaire (pour voir)
2
3 <<< resultat = requests.get('https://jsonplaceholder.typicode.com/posts/10')
4 <<< resultat
5 <Response [200]> # la requête s'est bien déroulée
6 <<< resultat.json()
7 { 'userId': 1, 'id': 10, 'title ': 'optio molestias id quia eum', 'body':
8     'quo et expedita modi cum officia vel magnidoloribus qui
9     repudiandaevero nisi sitquos veniam quod sed accusamus veritatis
10    error '}
11   <<< modification = {'body': 'mon commentaire est beaucoup plus lisible '}
12   <<< resultat = requests.patch('https://jsonplaceholder.typicode.com/posts/100', json=modification)
13
14   <<< resultat
15   <Response [200]> # la requête s'est bien déroulée
16   <<< resultat.json()
17 { 'userId': 10, 'id': 100, 'title ': 'at nam consequatur ea labore ea
18    harum', 'body': 'mon commentaire est beaucoup plus lisible '}
```



☞ De même, **supprimez** un commentaire, une photo, etc. sur JSONPlaceHolder avec DELETE
✓ Exercice d'entraînement pour vous familiariser avec les API au besoin.

2.3 Utiliser l'API REST de Github

Une **API**, nous l'avons vu, est un "élément" informatique d'un site web concue pour interagir avec d'autres programmes. Pour quérir des informations, ces programmes utilisent souvent **une adresse spécifique**. Cette requête particulière s'appelle un "**API call**" (pour garder l'anglissime). Les données collectées sont retournées dans un format "**pivot**" facilement manipulable comme le format **JSON** (JavaScript Object Notation), le **CSV** (Comma Separated Values) pour les plus importants, ou encore en **XML** (eXtended Markup Langage) ou dérivés. La plupart des applications qui ont recours à des données externes (comme les médias sociaux), recourent à l'utilisation de ces "**API calls**" (Interfaces de Programmation d'Application en français).

2.4 Un Appel API : une autre Approche

Nous allons nous appuyer sur l'utilisation de la plateforme Github, nous l'avons vu, qui permet aux programmeurs de collaborer à l'écriture de projets informatiques (cf TP 1). Dans cet exemple du cours, nous utiliserons **l'API de Github** pour interroger et collecter des informations sur les codeurs qui utilisent majoritairement le langage **Python** et nous allons visualiser la popularité relative de ceux-ci en utilisant la librairie **Plotly**.

Quand des utilisateurs apprécient un développement, un espace ou un code particulier, ils peuvent notifier au programmeur leur remerciement, adhésion ou suivi de leur projet en attribuant une étoile ("star"). Nous allons donc écrire un programme très simple en python pour télécharger automatiquement les projets les plus "étoilés" sur Github et produire une représentation graphique des résultats.

L'adresse générale (et la forme simple) pour requêter sur Github ressemble à :

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

Adresse de l'API de GitHub et Requête

```
1 <!-- A Copier-Coller dans le champ URL de votre navigateur  
-->  
2  
3 https://api.github.com/search/repositories?q=language:  
    python&sort=stars
```

Notez la fin de l'adresse (`?q=`) qui indique explicitement la requête ("query") envoyée (par la méthode GET (voir cours UOR)), ici "langage :python" suivi d'un ordre de tri (`sort=stars`), pour obtenir par ordre décroissant les projets les plus récompensés en premier.

A la date de la conception de ces notes de TP, nous obtenons l'affichage suivant :

Résultats en Format JSON dans le Navigateur

```
1 {  
2     "total_count": 8665788,  
3     "incomplete_results": true,  
4     "items": [  
5         {  
6             "id": 51117837,  
7             "node_id": "MDEwOlJlcG9zaXRvcnk1MTEzMzgNw==",  
8             "name": "models",  
9             "full_name": "tensorflow/models",  
10            "private": false,  
11            "owner": {  
12                "login": "tensorflow",  
13                "id": 15658638,  
14                "node_id": "MDEyOk9yZ2FuaXphdGlvbjE1NjU4NjM4",  
15                ...  
16            }  
17        }  
18    ]  
19}
```

Les résultats obtenus sont "humainement" lisibles au format JSON mais il est plus pratique de les traiter à l'aide de programmes de traitement informatique. Nous voyons que GitHub a trouvé 8 665 788 projets Python enregistrés ("total_count") et comme l'item "incomplete_results" est positionné sur "true", nous savons que les résultats obtenus sont incomplets, car l'API n'a pu intégralement faire aboutir notre requête : il y en a sans doute beaucoup plus...

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



A votre tour, essayez de deviner la requête suivante :

Une Nouvelle Requête

```
1 <!-- A Copier-Coller dans le champ URL de votre navigateur
-->
2
3 https://api.github.com/search/issues?q=windows+label:bug+
language:python+state:open&sort=created&order=asc
```

Là encore, à la date de ces notes, nous obtenons la réponse au format JSON suivant :

Résultats obtenus au Format JSON

```
1 {
2   "total_count": 10820,
3   "incomplete_results": false,
4   "items": [
5     {
6       "url": "https://api.github.com/repos/
fredrik-johansson/mpmath/issues/77",
7       "repository_url": "https://api.github.com/repos/
fredrik-johansson/mpmath",
8       "labels_url": "https://api.github.com/repos/
fredrik-johansson/mpmath/issues/77/labels{/name}",
9       "comments_url": "https://api.github.com/repos/
fredrik-johansson/mpmath/issues/77/comments",
10      "events_url": "https://api.github.com/repos/
fredrik-johansson/mpmath/issues/77/events",
11      "html_url": "https://github.com/fredrik-johansson/
mpmath/issues/77",
12      "id": 32125397,
13      "node_id": "MDU6SXNzdWUzMjEyNTM5Nw==",
14      "number": 77,
15      ...
...
```

Nous obtenons pour cette seconde requête un total de 10 820 dont les résultats ont été cette fois intégralement traités : "incomplete_results" : false. Notez que dans l'adresse nous avons utilisé "/issues" en lieu et place de "/repositories".

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Il est, par exemple, possible de rechercher :

des Requêtes Génériques à Compléter

```
1 <!-- dans les codes-sources -->
2 https://api.github.com/search/code?q= ...
3 <!-- dans les commits -->
4 https://api.github.com/search/commits?q= ...
5 <!-- dans les "questions/problèmes" rencontrés (issues) -->
6 https://api.github.com/search/issues?q= ...
7 <!-- dans les étiquettes (mots-clés attribués) (labels) -->
8 https://api.github.com/search/labels?q= ...
9 <!-- dans les espaces (ou entrepôts) (Repositories) -->
10 https://api.github.com/search/repositories?q= ...
11 <!-- dans les sujets (Topics) -->
12 https://api.github.com/search/topics?q= ...
13 <!-- parmi les noms d'utilisateurs (users) -->
14 https://api.github.com/search/users?q= ...
```

2.5 Requêter Automatiquement avec un Langage de Programmation : Python

Reprendons notre première requête et donnons forme au résultat à l'aide de la librairie "requests" et du langage python :

Requête traitée avec Python

```
1 import requests
2 # pip3 install requests (ou conda install requests) au besoin
3 url = "https://api.github.com/search/repositories ?q=language:python&sort=stars"
4
5 headers = {'Accept': 'application/vnd.github.v3+json'}
6 # la version des données : "application/vnd.github.v3+json" est recommandé
    # dans la documentation
7 # voir https://docs.github.com/en/rest/overview/media-types pour en savoir
    # plus (QR-Code)
8
9 r = requests.get(url, headers=headers)
10 print(f"Status de la requête : {r.status_code}")
11
12 # Nous sauvegardons la réponse du serveur au format JSON dans une variable :
13 resultat = r.json()
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



```
14 # Nous traitons le résultat :  
15 print(resultat.keys())
```

Les Formats de l'API (Media Types)



Après exécution du programme, nous obtenons :

Exécution du Code Python

```
# exécution du programme par python marequete.py  
  
Status de la requête : 200  
dict_keys(['total_count', 'incomplete_results', 'items'])
```

Nous retrouvons les trois attributs "clés" de notre variable "resultat" conformément au code affiché dans le navigateur initial. Notons que la structure d'une page au format JSON s'apparente à un type "**dictionnaire**" (voir la documentation python au besoin). Nous avons également affiché le statut de la page renvoyé par le serveur : 200, signifiant que la requête a abouti (successful response). Il est généralement admis de ne pas tenir compte du booléen attribué à la clé "incomplete_results" lors d'une requête simple. Pour une requête plus complexe, notre programme pourra tester si la valeur "false" a bien été attribuée à la clé.

Les Codes de réponse HTTP sur Mozilla



CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Rappel des trois Clés affichées dans notre navigateur

```
1  {
2      "total_count": 10820,
3      "incomplete_results": false ,
4      "items": [
5          {
6              ...
7              ...
8              ...
```

Continuons notre code Python de manière à connaître tous les intitulés des clés (ou items) sur lesquelles nous pourrons requêter :

Recherche des intitulés des items (clés)

```
1 import requests
2 url ="https://api.github.com/search/repositories ?q=language:python&sort=stars"
3
4 r = requests.get(url, headers=headers)
5 print(f"Status de la requête : {r.status_code}")
6
7 # Nous sauvegardons le JSON dans une variable :
8 resultat = r.json()
9
10 print(f"Nombre Total d'Entrepôts: {resultat ['total_count']} ")
11 # Nous explorons les informations contenues (de la clé 'items')
12 repos = resultat ['items']
13
14 print(f"Nombre d'Entrepôts retournés: {len(repos)} ")
15
16 # Nous allons examiner le premier entrepôt juste pour voir :
17 repo = repos[0]
18 print(f"\nNombre de clés (items) trouvé : {len(repo)} ")
19 for key in sorted(repo.keys()):
20     print(key)
```

Nous obtenons quelque chose qui ressemble à la sortie suivante :

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

Exécution du Code Python

```
# l'exécution du programme par python marequete2.py

Status de la requête : 200
Nombre Total d'Entrepôts: 9242101
Nombre d'Entrepôts retournés: 30

Nombre de clés (items) trouvé : 78
allow_forking
archive_url
archived
assignees_url
blobs_url
branches_url
clone_url
collaborators_url
comments_url
commits_url
...
...
topics
trees_url
updated_at
url
visibility
watchers
watchers_count
```

Quand nous regardons ces différentes "clés", nous obtenons tous les "types" (ou champs) d'information que nous pouvons collecter. Ce procédé est utile pour connaître toutes les clés sur lesquelles nous pouvons opérer. Bien évidemment, n'hésitez pas à consulter la documentation de l'API que vous utilisez. Nous allons afficher quelques items (clés ou champs) des trois premiers entrepôts :

Affichage de quelques éléments "clés : valeurs"

```
1 import requests
2
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



```
3 url ="https://api.github.com/search/repositories ?q=language:python&sort=stars"
4
5 headers = {'Accept': 'application/vnd.github.v3+json'}
6 r = requests.get(url, headers=headers)
7
8 print(f"Status de la requête : {r.status_code}")
9
10 resultat = r.json()
11 repos = resultat [ 'items' ]
12
13 print(f"Nombre d'Entrepôts retournés: {len(repos)}")
14
15 # Nous allons examiner quelques attributs des trois premiers entrepôts :
16 for i in range(3):
17     repo = repos[i]
18     print(f"\nEspace numéro {i+1}")
19     print(f"Nom: {repo['name']}")
20     print(f"Propriétaire : {repo['owner']['login']}")
21     print(f"Etoiles Attribuées: {repo['stargazers_count']}")
22     print(f"Adresse de l'Entrepôt: {repo['html_url']}")
23     print(f"Crée le: {repo['created_at']}")
24     print(f"Mise à Jour: {repo['updated_at']}")
25     print(f"Description: {repo['description']}
```

Et nous obtenons l'affichage suivant :

Exécution du Code Python

```
# L'exécution du programme par python marequete3.py
Status de la requête : 200
Nombre d'Entrepôts retournés: 30

Espace numéro 1
Nom: system-design-primer
Propriétaire: donnemartin
Etoiles Attribuées: 183937
Adresse de l'Entrepôt: https://github.com/donnemartin/system-
design-primer
Crée le: 2017-02-26T16:15:28Z
Mise à Jour: 2022-06-15T20:27:15Z
Description: Learn how to design large-scale systems. Prep
for the system design interview. Includes Anki flashcards
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Espace numéro 2

Nom: youtube-dl

Propriétaire: ytdl-org

Etoiles Attribuées: 110804

Adresse de l'Entrepôt: <https://github.com/ytdl-org/youtube-dl>

Crée le: 2010-10-31T14:35:07Z

Mise à Jour: 2022-06-15T20:44:09Z

Description: Command-line program to download videos from YouTube.com and other video sites

Espace numéro 3

Nom: thefuck

Propriétaire: nvbn

Etoiles Attribuées: 71667

Adresse de l'Entrepôt: <https://github.com/nvbn/thefuck>

Crée le: 2015-04-08T15:08:04Z

Mise à Jour: 2022-06-15T20:22:42Z

Description: Magnificent app which corrects your previous console command.

Remarque

Avec un "for repo in repos :" (à la place de for i in range(3) : repo = repos[i]) nous affichons toutes les informations contenues dans tous les entrepôts retournés par notre requête.

2.6 Connaître les Limites de l'API

La plupart des API ont des accès limités dans le temps : savoir combien de requêtes il est possible de lancer dans un intervalle de temps donné est très utile, car il permet de s'assurer que notre collecte d'informations se fera correctement.

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Adresse Gitub (pour afficher le "Rate Limit") dans notre navigateur

```
1 https://api.github.com/rate_limit
```

Notre navigateur nous affiche une page au format JSON :

Affichage du "Rate Limit" pour l'API de Github

```
1 {
2   "resources": {
3     "core": {
4       "limit":60,
5       "remaining":58,
6       "reset":1550385312,
7       "used":0,
8       "resource":"core"
9     },
10    ...
11   "search": {
12     "limit":10,
13     "remaining":8,
14     "reset":1550381772
15   }
16 }
```

Ces informations sont intéressantes car elles nous disent que nous avons la possibilité de lancer 10 requêtes par minute, qu'il nous en reste encore 8. La valeur de l'item "reset", affichée en temps "Unix" (Epoch Time), c'est-à-dire en nombre de secondes écoulées depuis le 1er janvier 1970, nous indique quand notre quota sera remis à sa valeur maximale.

Attention

Un grand nombre d'API(s) demande à l'utilisateur de s'enregistrer pour obtenir une clé d'API afin de pouvoir requêter. Pour l'instant (à la date de ces notes de cours) Github n'exige pas cet enregistrement. Cependant si nous nous enregistrons, notre limite pourrait être plus élevée...



2.7 Construire une Représentation Graphique des Résultats

Nous allons réaliser ensuite une représentation graphique de notre requête pour visualiser les projets Python les plus populaires sur Github. Pour ce faire, modifions légèrement notre programme :

Visualisation du Résultat de notre Requête avec Plotly

```
1 import requests
2 from plotly.graph_objs import Bar
3 from plotly import offline
4 # pip3 install plotly (ou conda install plotly) au besoin
5
6 url ="https://api.github.com/search/repositories?q=language:python&sort=stars"
7
8 headers = {'Accept': 'application/vnd.github.v3+json'}
9
10 r = requests.get(url, headers=headers)
11 print(f"Status de la requête : {r.status_code}")
12
13 resultat = r.json()
14
15 # Nous stockons les informations contenues dans la clé (items)
16 repos = resultat['items']
17
18 # Nous collections les valeurs des champs "name" et "stargazers_count"
19 repo_names, stars = [], []
20
21 for repo in repos:
22     repo_names.append(repo['name'])
23     stars.append(repo['stargazers_count'])
24
25 # Nous construisons notre graphique :
26
27 data = [
28     {'type': 'bar',
29      'x': repo_names,
30      'y': stars,
31  }]
32
33 mon_agencement = {
34     'title': 'les Projets Python les plus étoilés sur GitHub',
35     'xaxis': {'title': "Nom de l'Espace"},
36     'yaxis': {'title': "Nombre d'Etoiles"},
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

```
37 }  
38  
39 fig = {'data': data, 'layout': mon_agencement}  
40 offline.plot(fig, filename='python_repos.html')
```

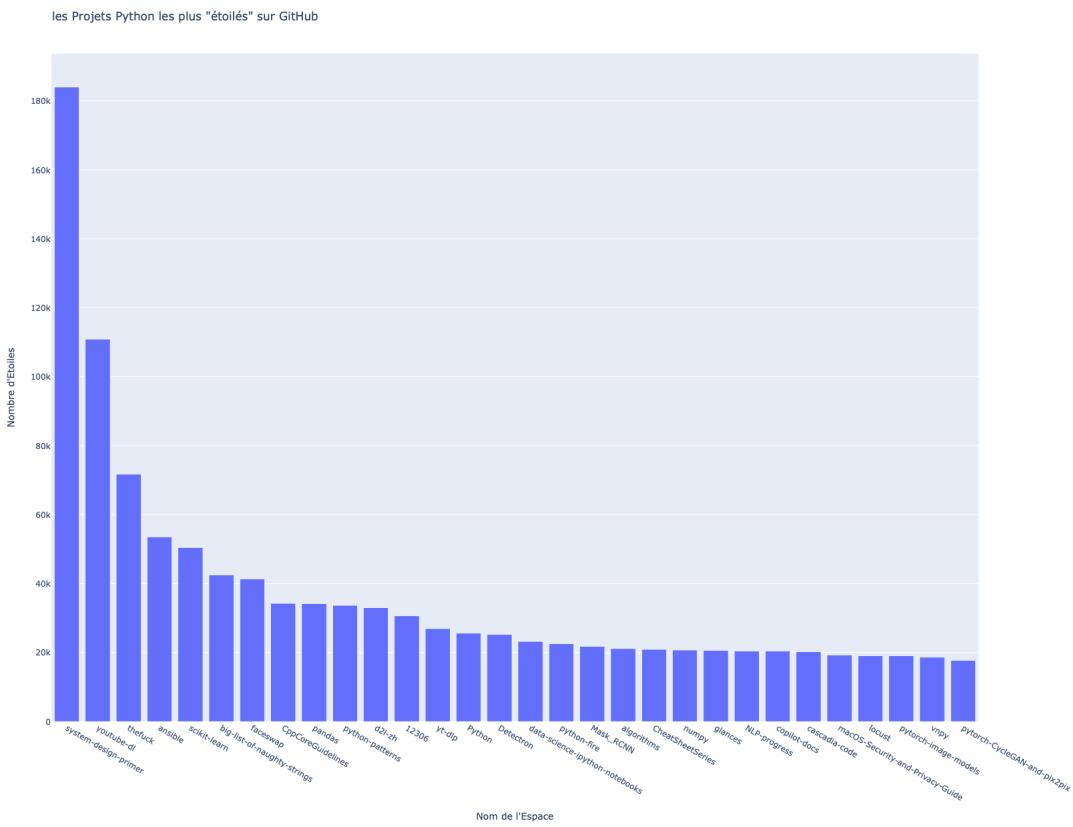
Nous avons importé la classe "Bar" et le module "offline" de la librairie plotly (ne pas oublier de l'installer au besoin). Voyez que nous avons continué à afficher le statut de notre requête (qui s'affiche dans notre terminal, notre IDE ou dans notre "notebook jupyter" selon).

La Documentation de la Librairie Plotly



A l'exécution de notre code Python, notre navigateur devrait afficher automatiquement la page produite par plotly (fig.1). Si ce n'est pas le cas, dans le même répertoire que notre code, ouvrons manuellement "python_repos.html" dans notre navigateur préféré. En survolant chaque élément de notre graphique avec notre souris, nous voyons s'afficher le nom de l'espace dans son intégralité, ainsi que le nombre d'étoiles obtenu. En haut à gauche, un menu spécial sous la forme de plusieurs icônes permet de réaliser des opérations diverses que nous pouvons tester au besoin (export au format png, zoom, comparaison, sélection, ...). Nous voyons que le graphique est une "zipfiennne", c'est-à-dire qu'elle suit une distribution "exponentielle" ici décroissante en forme significative de "coude", que nous pouvons retrouver dans les infométries de Mandelbrot, Benford, Lotka, ou Pareto. (Voir ces termes sur Wikipédia)

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



- Fig.2 - Affichage dans notre Navigateur Web -

De manière très facultative, nous pouvons améliorer légèrement l'apparence de notre graphique et son interactivité. Par exemple :

Amélioration de la Lisibilité de notre Graphique

```
1 import requests
2 from plotly.graph_objs import Bar
3 from plotly import offline
4
5 url = "https://api.github.com/search/repositories ?q=language:python&sort=stars"
6
7 headers = {'Accept': 'application/vnd.github.v3+json'}
8
9 r = requests.get(url, headers=headers)
10 print(f"Status de la requête : {r.status_code}")
11
12 resultat = r.json()
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

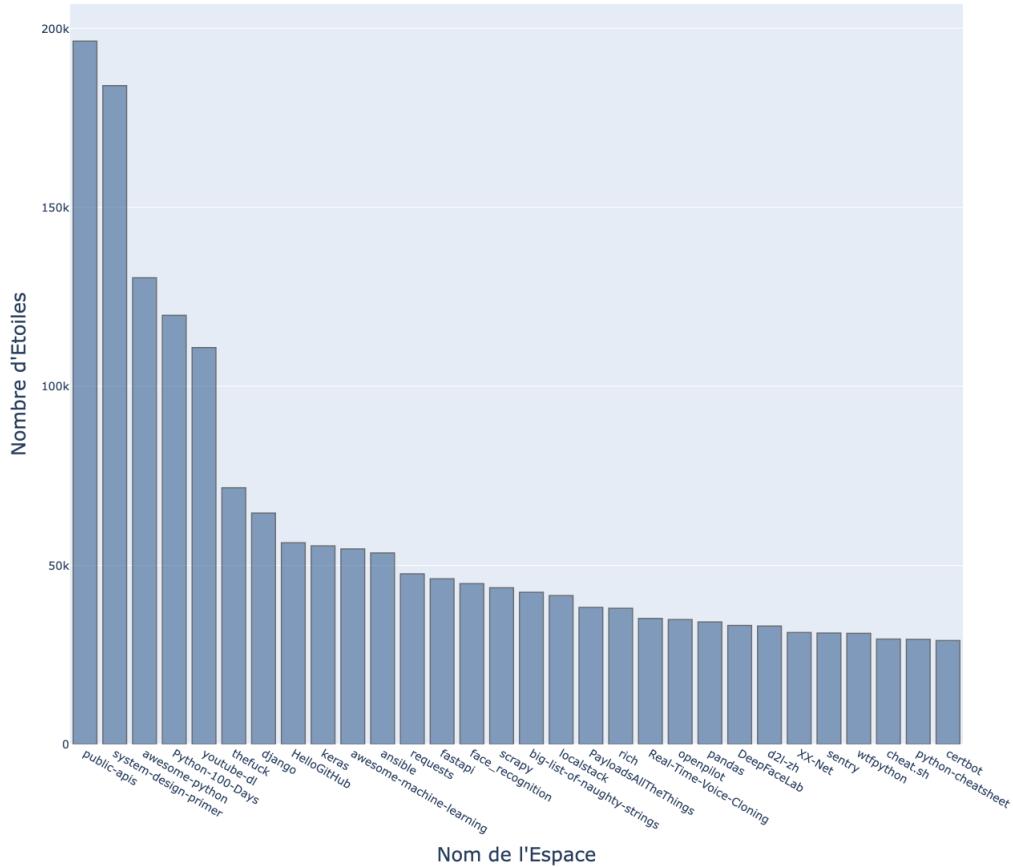


```
13
14 # Nous stockons les informations contenues dans la clé (items)
15 repos = resultat[ 'items' ]
16
17 # Nous collections les valeurs des champs "name" et "stargazers_count"
18 repo_names, stars = [], []
19
20 for repo in repos:
21     repo_names.append(repo['name'])
22     stars.append(repo['stargazers_count'])
23
24 # Nous construisons notre graphique :
25
26 data = [
27     'type': 'bar',
28     'x': repo_names,
29     'y': stars,
30 ]
31
32 mon_agencement = {
33     'title': 'les Projets Python les plus "étoilés" sur GitHub',
34     'xaxis': {'title': "Nom de l'Espace"},
35     'yaxis': {'title': "Nombre d'Etoiles"},
36 }
37
38 fig = {'data': data, 'layout': mon_agencement}
39 offline.plot(fig, filename='python_repos.html')
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



les Projets Python les plus "étoilés" sur GitHub



- Fig.3 - Affichage modifié dans notre Navigateur Web -

Améliorons une troisième fois notre graphique en affichant le nom, la description et le nombre d'étoiles en survolant nos éléments :

Amélioration des informations des "Tooltips"

```
1 import requests
2 from plotly.graph_objs import Bar
3 from plotly import offline
4
5 url = "https://api.github.com/search/repositories ?q=language:python&sort=stars"
6
7 headers = {'Accept': 'application/vnd.github.v3+json'}
8
9 r = requests.get(url, headers=headers)
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



```
10 print(f"Status de la requête : {r.status_code}")
11
12 # Nous sauvegardons le JSON dans une variable :
13 resultat = r.json()
14
15 # Nous stockons les informations contenues dans la clé (items)
16 repos = resultat['items']
17
18 # Nous collections les valeurs des champs "name", "stargazers_count", "owner"
19 # et "description"
20 repo_names, stars, labels = [], [], []
21
22 for repo in repos:
23     repo_names.append(repo['name'])
24     stars.append(repo['stargazers_count'])
25
26     owner = repo['owner']['login']
27     description = repo['description']
28     label = f"{owner}<br />{description}"
29     labels.append(label)
30
31 # Nous construisons notre graphique :
32
33 data = [
34     {
35         'type': 'bar',
36         'x': repo_names,
37         'y': stars,
38         'hovertext': labels, # ne pas oublier cette ligne
39         'marker': {
40             'color': 'rgb(60, 100, 150)',
41             'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'},
42             'opacity': 0.6,
43         }
44     }
45 ]
46
47 mon_agencement = {
48     'title': 'les Projets Python les plus "étoilés" sur GitHub',
49     'titlefont': {'size': 28},
50     'xaxis': {
51         'title': "Nom de l'Espace",
52         'titlefont': {'size': 24},
53         'tickfont': {'size': 14},
54     },
55     'yaxis': {
56         'title': "Nombre d'Etoiles",
57         'titlefont': {'size': 24},
58         'tickfont': {'size': 14},
59     },
60 }
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

```
55 }  
56  
57 fig = { 'data': data, 'layout': mon_agencement }  
58 offline.plot(fig, filename='python_repos.html')
```



- Fig.4 - Un "Tooltip" dans notre Navigateur Web -

Une dernière amélioration de l'interactivité de notre graphique en ajoutant la possibilité de cliquer sur le nom de chaque élément affiché en abscisse afin de se rendre dans la page principale de l'espace sélectionné en utilisant l'URL de celui-ci :

Amélioration de l'Interactivité (avec URL)

```
1 import requests  
2 from plotly.graph_objs import Bar  
3 from plotly import offline  
4  
5 url ="https://api.github.com/search/repositories ?q=language:python&sort=stars"  
6  
7 headers = {'Accept': 'application/vnd.github.v3+json'}  
8  
9 r = requests.get(url, headers=headers)  
10 print(f"Status de la requête : {r.status_code}")  
11
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



```
12 # Nous sauvegardons le JSON dans une variable :
13 resultat = r.json()
14
15 # Nous stockons les informations contenues dans la clé (items)
16 repos = resultat['items']
17
18 repo_links, stars, labels = [], [], [] # remplacement de repo_names par
19     repo_links
20
21 for repo in repos:
22     repo_name = repo['name']
23     repo_url = repo['html_url'] # changements et 2 lignes suivantes
24     repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
25     repo_links.append(repo_link)
26
27     stars.append(repo['stargazers_count'])
28
29     owner = repo['owner']['login']
30     description = repo['description']
31     label = f'{owner}<br />{description}'
32     labels.append(label)
33
34     # Nous construisons notre graphique :
35
36     data = [
37         {
38             'type': 'bar',
39             'x': repo_links, # liens ici en abscisse
40             'y': stars,
41             'hovertext': labels,
42             'marker': {
43                 'color': 'rgb(60, 100, 150)',
44                 'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'}, },
45                 'opacity': 0.6,
46             }
47
48     mon_agencement = {
49         'title': 'les Projets Python les plus "étoilés" sur GitHub',
50         'titlefont': {'size': 28},
51         'xaxis': {
52             'title': "Nom de l'Espace",
53             'titlefont': {'size': 24},
54             'tickfont': {'size': 14},
55         },
56         'yaxis': {
57             'title': "Nombre d'Etoiles",
58             'titlefont': {'size': 24},
59             'tickfont': {'size': 14},
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

```
57     },
58 }
59
60 fig = {'data': data, 'layout': mon_agencement}
61 offline .plot(fig , filename='python_repos.html')
```

Par exemple, en cliquant sur le quatrième élément de notre graphique, nous nous rendons dans la page principale de l'espace "Keras", une librairie dédiée à l'apprentissage profond (Deep Learning) bien connue des data-scientists.



- Fig.5 - La page principale du projet "Keras" -

2.8 Découvrir une seconde API : Hacker News

Hacker News est un site de partage collaboratif de liens créé en 2007 (Starto News initialement) crée par Paul Graham (Y Combinator), qu'il a développé avec langage de programmation Arc. IL a pour objectif de fonder une communauté centrée autour de l'actualité des technologies informatiques (IT). Les news sont présentées selon un algorithme de classement (date, taux de rebond, commentaires...) dont ces derniers se doivent d'être constructifs et non négatifs.

Pour explorer celui-ci, nous allons visiter ce site et lancer quelques requêtes anonymes (sans utiliser de clés d'API, mais rien de vous empêche de vous enregistrer et d'aller plus loin...).

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Adresse de l'API de Hacker News et Requête

```
1 <!-- A Copier-Coller dans le champ URL de votre navigateur  
-->  
2  
3 https://hacker-news.firebaseio.com/v0/item/28300951.json
```

Comparativement à la requête GitHub, nous requêtons ici directement sur le fichier JSON :

Résultat obtenu (JSON) dans le Navigateur

```
{"by": "sebg", "descendants": 74, "id": 28300951, "kids": [28305468, 28302681, 28303935, 28302599, 28309446, 28305668, 28305868, 28301542, 28305210, 28302191, 28302680, 28303578, 28303606, 28304055, 28303499, 28318334, 28302644, 28307229, 28303894, 28303078, 28302462, 28303059, 28303292, 28305858, 28305312, 28304976, 28302695, 28307533, 28303294, 28306662], "score": 982, "time": 1629895706, "title": "Prettymaps: Small Python library to draw customized maps from OpenStreetMap data", "type": "story", "url": "https://github.com/marceloprates/prettymaps"}
```

Comme nous pouvons le voir, ce format JSON fonctionne un peu comme tous les JSON, c'est-à-dire selon le type d'un dictionnaire. Cependant, la réponse est difficile à lire et nous allons, à l'aide de Python, formatter cet article (cette sortie) :

Amélioration de la Présentation de la Réponse

```
1 import requests  
2 import json  
3  
4 # Un Appel API à Hacker News, affichage et sauvegarde de la réponse  
5 url = 'https://hacker-news.firebaseio.com/v0/item/28300951.json'  
6 r = requests.get(url)  
7 print(f"Status code: {r.status_code}") # important pour test  
8
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

```
9 # Affichage formaté de la réponse :  
10 reponse = r.json()  
11 print(json.dumps(reponse,sort_keys=False, indent=4))
```

Ce qui nous donne après avoir sauvegardé et lancé notre code dans le Terminal :

Résultat en JSON Formaté après Execution du Code Python

```
Status code: 200  
{  
    "by": "sebg",  
    "descendants": 74,  
    "id": 28300951,  
    "kids": [  
        28305468,  
        28302681,  
        28303935,  
        ...  
        28306662  
    ],  
    "score": 982,  
    "time": 1629895706,  
    "title": "Prettymaps: Small Python library to draw  
customized maps from OpenStreetMap data",  
    "type": "story",  
    "url": "https://github.com/marceloprates/prettymaps"  
}
```

La sortie du programme affiche bien les métadonnées (sous la forme d'un dictionnaire) d'un article dont l'identifiant (ID) est 28300951 et pour lequel nous apercevons plusieurs items sous la forme d'un couple (clés et valeurs) : le titre, le type, l'adresse, le score obtenu, etc. La liste "kids" comporte tous les ID(s) des commentaires reçus par cet article et "descendants" nous donne le nombre de ceux-ci. Par souci de lisibilité, la liste a été réduite ici.

Requêtons une nouvelle fois, mais en recherchant les articles les plus lus :

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Adresse de l'API de Hacker News et Requête

```
1 <!-- A Copier-Coller dans le champ URL de votre navigateur  
-->  
2  
3 https://hacker-news.firebaseio.com/v0/topstories.json
```

Comme vous le voyez, le résultat affiché par votre navigateur est encore moins "humainement" lisible puisque nous obtenons une liste d'ID(s) :

Résultat obtenu (Liste d'ID(s)) dans le Navigateur

```
[31810104, 31808093, 31810511, 31790246, 31810832,  
31807783, 31807913, 31807201, 31811341, 31808844, ...,  
31756643, 31769520, 31794175]
```

Pour obtenir des informations sur cette liste d'identifiants (simplifiée ici), nous allons requêter Hacker News, individuellement pour chaque ID :

Amélioration de la Présentation de la Réponse

```
1 import requests  
2 from operator import itemgetter  
3  
4 # Un Appel API à Hacker News (Top Stories), affichage et sauvegarde de la réponse  
5 url = 'https://hacker-news.firebaseio.com/v0/topstories.json'  
6 r = requests.get(url)  
7 print(f"Status code: {r.status_code}")  
8  
9 # Requête pour chaque identifiant contenu dans la liste (r) :  
10 ids = r.json()  
11 information_dict = []  
12 for id in ids [:10]:  
13     # Nous réalisons un appel API pour chaque ID, nous nous sommes limités aux  
14     # 10 premiers :  
15     # écrire : for id in ids : (sinon ... attention, ça peut être plus long )  
16     url = f"https://hacker-news.firebaseio.com/v0/item/{id}.json"  
17     r = requests.get(url)  
18
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



```
19     # Nous affichons le code de statut (200) -- à supprimer ensuite
20     print(f"id: {id}      status: {r.status_code}")
21     reponse= r.json()
22     # Nous construisons un dictionnaire pour stocker les informations qui nous intéressent :
23     # Un premier test nous a montré que certaines clés sont quelquefois absentes
24     # des enregistrements (c'est le cas de "descendants" ici).
25     # Nous adaptons :
26
26     if 'descendants' not in reponse:
27         reponse['descendants'] = ''
28
29     information = {
30         'titre': reponse['title'],
31         'url': f"http://news.ycombinator.com/item?id={id}",
32         'commentaires': reponse['descendants'],
33     }
34     information_dict.append(information)
35
36
37     mes_articles = sorted(information_dict, key=itemgetter('commentaires'), reverse
38                           =True)
39
39     for article in mes_articles:
40         print(f"\n Titre : {article ['titre']}")
41         print(f"Adresse : {article ['url']}")
42         print(f"Commentaires: {article ['commentaires']}")
```

Comme à l'habituée, nous réalisons notre appel API et nous affichons le statut de la réponse. Cet appel renvoie une liste contenant tous les identifiants (ID) des 500 articles les plus populaires à la date de notre requête. Par souci de clarté et de temps, nous nous sommes limités à l'affichage des 10 premiers (vous pouvez bien évidemment changer cette valeur). Chaque contribution sur Hacker News est évaluée selon plusieurs critères : combien de commentaires celui-ci a-t-il reçu, combien de votes favorables, la date d'envoi, etc. Ainsi nous avons organisé notre liste d'articles selon le nombre décroissant de commentaires, à l'aide de la fonction "itemgetter()".

Après sauvegarde et exécution de notre code python dans le Terminal :

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE

Résultats et Traitements obtenus par notre Requête

```
Status code: 200
id: 31810104    status: 200
id: 31808093    status: 200
id: 31810511    status: 200
...
id: 31808866    status: 200
```

Titre: Facebook says Apple is too powerful - They're right ?
Adresse : <http://news.ycombinator.com/item?id=31807783>
Commentaires: 320

Titre: Writing One Sentence per Line
Adresse : <http://news.ycombinator.com/item?id=31808093>
Commentaires: 179

Titre: Stolperstein
Adresse : <http://news.ycombinator.com/item?id=31790246>
Commentaires: 147

Titre: We are removing the option to create new subscriptions
Adresse : <http://news.ycombinator.com/item?id=31810104>
Commentaires: 131

Titre: The integrated timetable of Switzerland
Adresse : <http://news.ycombinator.com/item?id=31807913>
Commentaires: 114

Titre: Silero V3: fast high-quality text-to-speech in 20 languages with 173 voices
Adresse : <http://news.ycombinator.com/item?id=31807201>
Commentaires: 70

Titre: SSO should be table stakes
Adresse : <http://news.ycombinator.com/item?id=31810832>
Commentaires: 49

Titre: Owncast is a self-hosted live video and web chat server
Adresse : <http://news.ycombinator.com/item?id=31810511>
Commentaires: 21

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Titre: The case for unique email addresses (2020)
Adresse : <http://news.ycombinator.com/item?id=31811341>
Commentaires: 10

Titre: Sega Megadrive Synthesizer in Depth Look (2020) [video]
Adresse : <http://news.ycombinator.com/item?id=31808866>
Commentaires: 0

Nous pouvons ainsi accéder et analyser **tout type d'information avec n'importe quelle API**. Avec les données que nous avons collectées, nous pourrions réaliser une représentation graphique des résultats, afficher les thématiques ou les termes les plus fréquents (cf GIL, chapitre 1), etc. Le plus simple est de toujours consulter la **documentation de l'API** que nous souhaitons utiliser.

La Documentation de l'API Hacker News sur Github



2.9 Jouer avec une troisième API : Giphy

Giphy est un *site graphique collaboratif*, créé en 2013 par Alex Chung et Jace Cooke, et contenant une base de données permettant la recherche, le partage et la création de GIF animés et d'auto-collants, ceux-ci ayant la particularité d'être dépourvus de sons. Giphy est devenu la propriété de la société de M. Zuckerberg, Facebook puis Meta en 2020.

A la différence des API de Github et Hacker News, Giphy nécessite que nous nous enregistrons et que nous récupérons **une clé d'API** (API key) avant de lancer une recherche.

- Pour ce faire, rendons-nous sur le site de **Giphy** pour nous enregistrer (gratuitement) avec notre mail et un mot de passe de notre choix.

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



- Il suffit ensuite, après avoir saisi les quatre lettres demandées et reçues par mail en retour, de nous rendre dans l'espace **développeur** et de créer une nouvelle application (choisissez bien API et non SDK). Nous pouvons saisir dans la formulaire qui s'affiche n'importe quoi comme *nom* et comme *description*, cela n'a pas d'importance. En cas de difficulté, vous pouvez consulter *la documentation ici*.
- Nous pouvons ensuite, dans notre "tableau de bord" (Dashboard) récupérer notre clé d'API par copier-coller et l'utiliser dans un premier temps avec **l'explorateur d'API en ligne** :

The screenshot shows the Giphy API Explorer interface. At the top, there's a navigation bar with links for Docs, Dashboard, FAQs, API Explorer, Blog, and Create. Below the navigation is the title "API Explorer". A sub-instruction "Take our API for a spin by inputting some sample queries and view live responses!" follows.

Request section:

- "Choose an app / API Key": A dropdown menu with "Essai d'utilisation de l'API" selected, and a red placeholder "votre clé" is shown.
- "Choose a resource": A dropdown menu with "GIPHY Public API" selected.
- "Choose an endpoint": A dropdown menu with "Trending" selected.
- "Request URL": A text input field containing the URL `https://api.giphy.com/v1/gifs/trending?api_key= votre clé &limit=25&rating=g`.
- A blue "Send Request" button.

Parameters section:

- "limit": Set to 25.
- "rating": Set to g.

Response section:

```
{  
  "data": [  
    {  
      "type": "gif",  
      "id": "3GtrY0mcGn2mMGDRFg",  
      "url": "https://giphy.com/gifs/neonrated-neon-spencer-rated-3GtrY0mcGn2mMGDRFg",  
      "slug": "neonrated-neon-spencer-rated-3GtrY0mcGn2mMGDRFg"  
    }  
  ]  
}
```

- Fig.6 - L'Explorateur API en ligne de Giphy -

Nous voyons dans le champ "*response*" les informations collectées dans

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



le format JSON que nous commençons à bien maîtriser. Utilisons comme précédemment Python pour organiser celles-ci et ne garder que les informations qui nous intéressent :

Collecte et Organisation des Informations avec l'API Giphy

```
1 import requests
2
3 # Remplacez <votre clé API> par celle que vous avez générée :
4 API_KEY = "<votre clé API>"
5 url = "https://api.giphy.com/v1/gifs/trending"
6
7 # Ici nous récupérons les 10 premiers 'gifs' (changez cette valeur au besoin)
8 params = {"api_key": API_KEY, "limit": 10, "rating": "g"}
9
10 reponse = requests.get(url, params=params).json()
11
12 # boucle sur la réponse
13 for gif in reponse["data"]:
14     title = gif[ "title "]
15     trending_date = gif[ "trending_datetime"]
16     url = gif[ "url "]
17     print(f"\n{title} \n{trending_date} \n{url}\n")
```

Vous devez obtenir une sortie semblable à celle-ci :

Résultats

```
Kristen Stewart Neon Rated GIF by NEON
2022-06-24 08:00:12
https://giphy.com/gifs/neonrated-neon-spencer-rated-3
GtrY0mcGn2mMGDRFg

Celebrate Happy Birthday GIF by Loly in the sky
2018-10-04 22:30:01
https://giphy.com/gifs/lolyinthesky-party-happy-birthday-
miauricio-dSdoQogGb1Knid5Wb3

Im Out Season 8 GIF by The Office
2022-06-24 07:17:37
https://giphy.com/gifs/theoffice-the-office-tv-episode-806-
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



aJP6pxwPh5J1Hgvrpb

Happy Good Morning GIF

0000-00-00 00:00:00

<https://giphy.com/gifs/bird-eleonoraanimation-eleonorakuz-11ANeu88cEAsq8sD09>

I'm Out Season 8 GIF by The Office

2022-06-24 07:17:37

<https://giphy.com/gifs/theoffice-the-office-tv-episode-806-aJP6pxwPh5J1Hgvrpb>

Checking In I Love You GIF by Seize the Awkward

2022-05-29 21:28:04

<https://giphy.com/gifs/seizetheawkward-mental-health-how-are-you-seize-the-awkward-gHKnFHkGPUmG4GTzUt>

Sweet Dreams GIF

0000-00-00 00:00:00

<https://giphy.com/gifs/yellow-jMRB9vZZVstbi>

Cat Celebrate GIF

0000-00-00 00:00:00

<https://giphy.com/gifs/party-celebrate-birthday-DFexVkJRG7gX9oCy68r>

I Love Hearts GIF

2021-02-13 19:15:05

<https://giphy.com/gifs/animation-2d-Z21HJj2kz9uBG>

Happy Birthday Dance GIF by Deadlyie

0000-00-00 00:00:00

<https://giphy.com/gifs/deadlyie-birthday-cake-hbd-31VVbt1NFZ2B2Et8hY>

En changeant l'adresse de l'API, nous pouvons également rechercher des images animées qui ont été indexées avec un ou plusieurs mot-clé(s) en particulier (ici "computer") :

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Collecte et Extractions d'Images relativement à un Mot-Clé

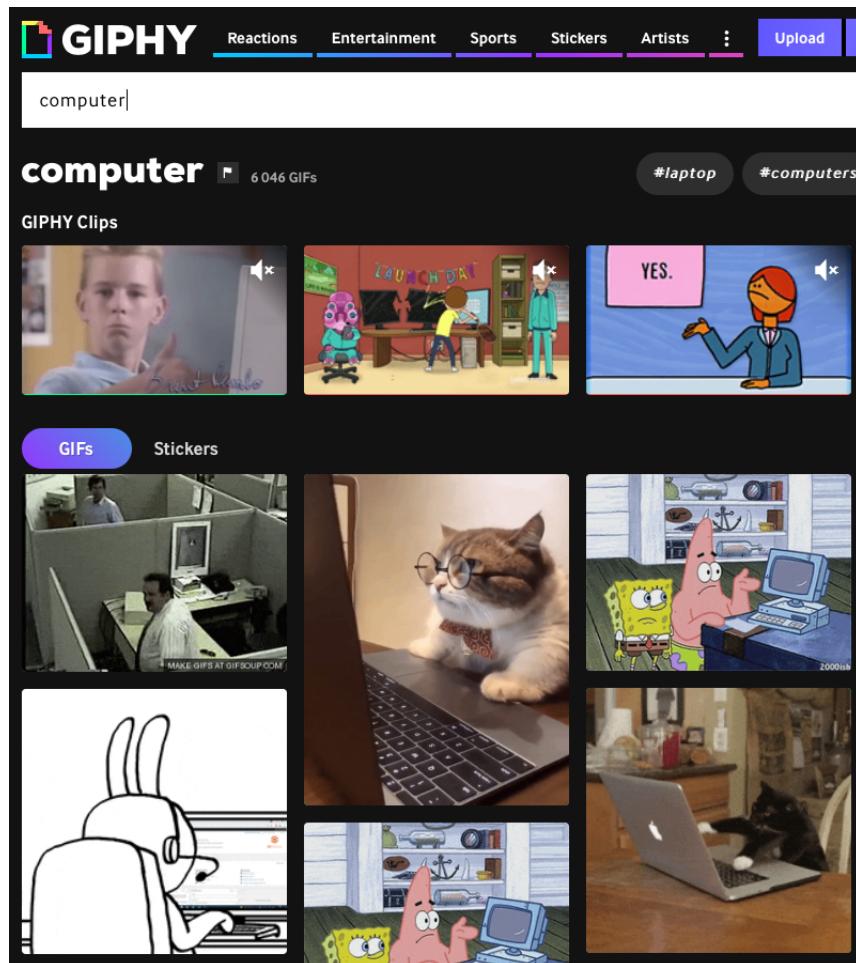
```
1 import requests
2
3 # Remplacez <votre clé API> par celle que vous avez générée :
4 API_KEY = "<votre clé API>"
5
6 # adresse de recherche
7 url = "https://api.giphy.com/v1/gifs/search"
8
9 # A changer selon (search_term, limit, rating,...)
10 search_term = "computer"
11 params = {"api_key": API_KEY, "limit": 10, "q": search_term, "rating": "g"}
12 reponse = requests.get(url, params=params).json()
13
14 # boucle sur la réponse (informations collectées ici juste le titre )
15 for gif in reponse["data"]:
16     title = gif[ "title "]
17     print(f "{title }")
```

Et nous obtenons après sauvegarde et exécution (à la date de la requête) :

Résultats (seulement les titres pour notre exemple)

```
Mad The Internet GIF by MOODMAN
Work From Home Reaction GIF
Cat Reaction GIF
spongebob squarepants computer GIF
Computer Working GIF
confused the simpsons GIF
Triple H Reaction GIF by WWE
3Rd Rock From The Sun Computer GIF
Hacking Work From Home GIF
Computer Working Late GIF by NOHARA
```

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



- Fig.7 - L'Interface de Giphy -

La Documentation de l'API Giphy





Les API en Quelques Mots :

Une API agit comme un médiateur entre les utilisateurs ou clients et les ressources ou services web auxquels ils souhaitent accéder. C'est aussi une solution pour partager collaborativement des données et des informations, tout en maintenant un certain niveau de sécurité, de contrôle et d'authentification.

L'API REST est un style architectural qui permet aux programmes de communiquer entre eux, quels que soient les systèmes d'exploitation. Les API REST fonctionnent sur la base d'une relation client-serveur en fournissant une interface uniforme pour la collecte des données. Elles sont dites auto-descriptives, multicouches et utilisent la contrainte HATEOAS (Hypermedia as the Engine of Application State) (cf Wikipédia au besoin) ce qui signifie que la réponse de l'API comprend des données qui donnent accès à d'autres méthodes disponibles. Elles sont de ce fait "sans état", c'est-à-dire que le serveur ne maintient pas de connexions ou de sessions entre les appels (voir cours L2-Réseaux). Ainsi les fonctionnalités essentielles des API sont une haute disponibilité et des réponses très rapides, facilement traitables par les langages et outils informatiques.

2.10 Les Exercices du TP

Pour ce second chapitre, vous devez ajouter à votre PDF l'**exercice 2.2**.
L'exercice 2.3 est pour un Bonus.

Exercice 2.2 (A Rendre)

En utilisant une ou plusieurs API de votre choix, comme par exemple, en choisissant parmi cette liste de liens ([Google Books](#), [Any API](#), [Public API](#), [API Gouv \(Service Public\)](#) [Open Data](#), [Open Food Facts](#), ...), ou celles présentées dans ces notes ou autres :

☞ **Réalisez à minima deux requêtes différentes** (voire plus) afin de collecter de l'information que vous préciserez : (Quelle(s) information(s) ? Pour quoi faire ?, Comment ?, Quel(s) traitements ?, etc.).

☞ Produisez une ou plusieurs représentations graphiques des résultats, à l'instar du cours, en n'oubliant pas d'indiquer vos motiva-



tions et d'en analyser les tendances et signaux remarquables.

- ✓ Pour les étudiant.e.s qui auraient déjà réalisé les TP(s) GIL, vous pouvez utiliser Tropes, Gephi ou encore des libraires de TAL (NLTK, TextBlob, SpaCy, GenSim, ...) , ou tout autre outil pour réaliser une cartographie sémantique (graphe, nuage de mots, ...) ou une analyse de sentiments à l'aide de l'information collectée.
- ✓ Les exemples ont été donnés en Python. Vous pouvez utiliser ce langage ou tout autre de votre choix. Renseignez bien les commandes et librairies utilisées et les directives de compilation si nécessaire.
- ✓ N'oubliez pas, comme à l'habituée, d'expliquer votre méthodologie, d'insérer et de fournir les codes commentées avec métadonnées descriptives (+ à joindre en archive attachée) et d'insérer des C.E(s) illustratives.

Exercice 2.3 (Bonus)

A l'aide de la Librairie Python **Selenium** et avec l'API et la (les) requêtes de votre choix :

- ☞ Concevez un programme qui collecte une dizaine d'adresses (URL) environ et qui génère dynamiquement les copies d'écran des sites web relativement à ces adresses.
- ☞ Affichez ces copies d'écran avec les métadonnées de votre choix (adresse (URL), titre, ...) dans une page web à l'aide de **Pyscript**.
- ✓ Les étudiant.e.s qui auraient quelques difficultés avec Pyscript peuvent utiliser et rendre un **Jupyter Notebook** à la place. Soignez la présentation et l'organisation de celui-ci en n'oubliant pas de le joindre à l'archive.
- ✓ Il est également possible d'utiliser la librairie **Pillow** (ex **PIL**) pour afficher les images dans un GUI de votre choix...
- ✓ Pensez à bien expliquer vos choix, vos méthodes et à bien citer les sources utilisées.

Pour celles et ceux qui s'intéressent à l'actualité sanitaire, il est possible, pour l'**exercice 2.2** d'utiliser l'API (gratuite) du Covid-19 sur **Postman**, qui sert à exécuter des appels HTTP directement depuis une interface graphique (ou par programmation) :

CHAPITRE 2. API - COLLECTER DE L'INFORMATION DE MANIÈRE COLLABORATIVE



Exemple d'utilisation de l'API du Covid-19

Covid API

```

1 import requests
2
3 from datetime import date, timedelta
4
5 auj = date.today()
6 hier = auj - timedelta(days=1)
7 pays = "france"
8 url = f"https://api.covid19api.com/country/{pays}/status/confirmed"
9 parametre = {"from": str(hier), "to": str(auj)}
10
11 response = requests.get(url, params=parametre).json()
12 total = 0
13 for day in response:
14     cas = day.get("Cases", 0)
15     total += cas
16
17 print(f"Le nombre total de Cas de Covid-19 confirmés pour le pays {pays}
      est de : {total }")

```

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of API endpoints under the 'CORONAVIRUS COVID19 API' collection. The 'By Country' endpoint is selected. The main panel displays the 'GET By Country' request details. It includes the URL, a description of returning all cases by country, parameters for 'from' and 'to' dates, and an example of the JSON response body.

- Fig.8 - L'Interface Postman de l'API COVID 19 -



2.11 Quelques Liens pour Aller plus Loin...

Quelques Sites et Livres Complémentaires à Explorer...

- (1) Un extrait en ligne de l'ouvrage "les API pour les Nuls" de Claus T. Jensen (First,2015) :



- (2) Les meilleures librairies NLP en Python (2020) :



- (3) Des mots aux sacs de mots (bag of words). Beaucoup de choses intéressantes sur ce site (à voir) :



Troisième partie

TP 3 - Créer sa Propre API

Chapitre 3

Développer une RESTful API avec FastAPI

DANS ce troisième TP, nous allons créer notre propre API et proposer de créer quelques requêtes simples. Nous verrons ce qu'est FastAPI et comment il se rapporte aux autres technologies Web dans l'écosystème de Python. Rappelons-nous que REST est un acronyme qui signifie *REpresentation State Transfer*, représentation développée par Roy Fielding en 2000. Une API REST utilise HTTP et les méthodes HTTP existantes pour implémenter les quatre opérations CRUD standard d'une base de données. REST mappe les méthodes HTTP (POST, GET, PUT et DELETE) respectivement aux fonctions CRUD (create, read, update et delete). Cette technologie réutilise également les codes d'état HTTP (200, 201, 404,...). toute API REST est accessible à l'aide d'un point de terminaison HTTP (entry point), conceptuellement une adresse Web et rend le résultat d'une requête principalement en format JSON.

3.1 Qu'est-ce FastAPI?

FastAPI facilite la création d'API Web REST. Il est simple et rapide, rivalisant avec d'autres frameworks Web, tels que Django RESTful, Flask RESTful, node.js et Go. Elle s'appuie sur des normes telles que OpenAPI et JSON Schema et sur les dernières fonctionnalités de Python comme l'indication de type (> 3.6). Elle fournit même un support asynchrone, sans avoir à utiliser la délicate bibliothèque Python asyncio.

Cet outil a été créé par Sebastian Ramírez (Tiangolo) en 2018 car ce dernier n'était pas satisfait par les frameworks existants comme Flask et Django



RESTful. C'est la raison pour laquelle il a créé son propre framework à l'aide d'outils comme *Starlette* et *Pydantic*. Aujourd'hui, de nombreuses entreprises comme Uber, Netflix et Microsoft utilisent FastAPI pour créer leurs applications. Conçu pour être simple à apprendre et à utiliser, FastAPI délivre un code prêt pour la production avec une documentation interactive automatique.

3.2 Installation de FastAPI

Avant de nous lancer dans l'installation de FastAPI, nous allons vérifier que notre version de Python est bien supérieure ou égale à la version 3.6, en saisissant dans notre Terminal :

Python >= 3.6?

```
python3 --version  
# ou encore python --version  
  
Python 3.8.3 # par exemple ici
```

Une fois rassuré.e, il suffit d'installer fastAPI (et le serveur associé uvicorn) avec l'utilitaire PIP :

Installation de fastAPI (2 méthodes)

```
pip install "fastapi[all]"  
# ou encore (selon)  
pip install fastapi uvicorn[standard]
```

Tutoriel d'Installation de FastAPI sur Tiangolo



Pour lancer nos requêtes ou pour voir les résultats, nous pouvons utiliser :



Postman, un outil graphique pour effectuer des requêtes HTTP ou encore **cURL**, l'outil de ligne de commande bien connu et largement utilisé pour effectuer des requêtes réseau.

Postman et cURL sur Wikipédia



Même si, de manière générale, les outils visuels sont agréables et faciles à utiliser, ils manquent parfois de flexibilité et peuvent ne pas être aussi productifs que les outils en ligne de commande. Nous pourrions utiliser **cURL** qui est un outil très puissant avec des milliers d'options mais celui-ci peut devenir complexe et verbeux pour tester des API REST simples. C'est pourquoi nous allons utiliser la **librairie HTTPIE**, un outil de ligne de commande visant à effectuer des requêtes HTTP avec une syntaxe intuitive, un support JSON et une coloration syntaxique (avec la librairie colorama).

Installations des Librairies HTTPIE et Colorama (plusieurs solutions)

```
pip install httpie colorama
# ou encore (selon)
pip install httpie
pip install colorama

# pour les utilisateurs de MacOS (avec brew)
brew install httpie
# pour les utilisateurs de Linux (Ubuntu par ex)
sudo apt-get install httpie
```

Nous pouvons maintenant tester une API de notre choix, par exemple ici, celle fournie par Amelie (Assurance Maladie) :

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



Test de l'API datavaccin avec HTTPPie

```
1
2 http GET "https://datavaccin-covid.ameli.fr/api/v2/catalog/
  datasets?limit=10&offset=0&timezone=UTC"
3 # ou tout autre API de votre choix
```

Grâce à la librairie, l'affichage est en couleur (pas comme ci-dessous :-):

Résultat de la Commande http GET

```
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Headers: Authorization,
  X-Requested-With, Origin, ODS-API-Analytics-App,
  ODS-API-Analytics-Embed-Type,
  ODS-API-Analytics-Embed-Referrer, ODS-Widgets-Version,
  Accept
3 Access-Control-Allow-Methods: POST, GET, OPTIONS
4 Access-Control-Allow-Origin: *
5 Access-Control-Max-Age: 1000
6 Cache-Control: public, max-age=300
7 Connection: keep-alive
8 Content-Encoding: gzip
9 Content-Language: fr
10 ...
11 {
12   "datasets": [
13     {
14       "dataset": {
15         "attachments": [],
16         "data_visible": true,
17         "dataset_id": "donnees-de-vaccination-par-epci",
18         "dataset_uid": "da_3codpk",
19         "features": [
20           "analyze",
21           "timeserie"
22         ],
23         "fields": [
24           ...
25         ]
```



Installation et Documentation de la librairie HTTPPie



Un autre exemple est fourni par la documentation :

Un autre exemple avec HTTPPie

```

1 http GET "https://httpie.io/hello"
2 # les guillemets doubles sont facultatifs
3

```

Nous rend :

Résultat de la Commande avec coloration syntaxique et émoticônes

```

1 HTTP/1.1 200 OK
2 Connection: keep-alive
3 Content-Type: application/json; charset=utf-8
4 age: 0
5 cache-control: public, max-age=0, must-revalidate
6 content-length: 264
7 date: Mon, 18 Jul 2022 15:25:25 GMT
8 etag: "108-yw+Xn8xsGAsJIUMQvDLN7gAcQuC"
9 server: Vercel
10 strict-transport-security: max-age=63072000
11 x-matched-path: /api/hello
12 x-vercel-cache: MISS
13 x-vercel-id: fra1::iad1::gtcnn-1658157924271-c0d6e26346fe
14
15 {
16     "ahoy": [
17         "Hello, World! Thank you for trying out HTTPPie",
18         "We hope this will become a friendship."
19     ],
20     "links": {
21         "discord": "https://httpie.io/discord",
22         "github": "https://github.com/httpie",

```



```

23     "homepage": "https://httpie.io",
24     "twitter": "https://twitter.com/httpie"
25   }
26 }
```

Toujours en reprenant les exemples fournis dans le tutoriel (cf QR-Code) :

Un exemple de la méthode PUT avec HTTPie sur httpbin.org

```

1
2 http PUT pie.dev/put name=Philippe
3 # ou le prénom de votre choix
```

Nous rend :

Résultat de la commande PUT

```

1 HTTP/1.1 200 OK
2 Access-Control-Allow-Credentials: true
3 Access-Control-Allow-Origin: *
4 CF-Cache-Status: DYNAMIC
5 CF-RAY: 72cc450ab96b9bd4-FRA
6 Connection: keep-alive
7 Content-Encoding: gzip
8 Content-Type: application/json
9 ...
10 Server: cloudflare
11 Transfer-Encoding: chunked
12 alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400
13
14 {
15   "args": {},
16   "data": "{\"name\": \"Philippe\"}",
17   "files": {},
18   "form": {},
19   "headers": {
20     "Accept": "application/json, */*;q=0.5",
21     "Accept-Encoding": "gzip",
22     "Cdn-Loop": "cloudflare",
23     "Cf-Connecting-Ip": "2a01:cb11:3f7:9200:10d9:32dc
:6285:b565",
```



```

24     "Cf-Ipcountry": "FR",
25     ... / ...
26   },
27   "json": {
28     "name": "Philippe"
29   },
30   "origin": "2a01:cb11:3f7:9200:10d9:32dc:6285:b565",
31   "url": "http://pie.dev/put"
32 }
```

De même, il nous est possible de tester la méthode POST avec :

Un exemple de la méthode POST avec HTTPie sur httpbin.org

```

1 http POST pie.dev/post hello="Bonjour les étudiant.e.s de l
  'IED"
2 ... / ...
3 "json": {
4   "hello": "Bonjour les étudiants de l'IED"
5 }
6 }
```

Utilisation

Je vous invite à regarder la documentation de cet outil fort utile et à tester interactivement les exemples. Cette librairie pourra être utilisée à tout moment pour toute requête réseau, dans ce cours ou un autre, en local ou sur le web.

3.3 Quelques Informations (Facultatives) sur le Traitement des E/S Asynchrones

Si vous avez déjà programmé avec JavaScript et Node.js, vous avez certainement rencontré les mots clés *async/await*, caractéristiques du paradigme des I/O asynchrones. Fondamentalement, c'est un moyen très utile de rendre les opérations d'entrée-sorties (E/S) non bloquantes et de permettre au programme d'effectuer d'autres tâches pendant que l'opération



de lecture ou d'écriture est en cours. La principale motivation est que les opérations d'E/S sont relativement lente : la lecture à partir du disque, les requêtes réseau sont un million de fois moins rapides que la lecture à partir de la RAM ou le traitement des instructions avec le CPU.

Async et Await en JavaScript



Exemple en Python de Lecture (lente) d'un Fichier

```

1 with open("memoire.txt", 'r') as f:
2     mon_memoire_de_licence = f.read()
3 # Le programme s'arrête tant que "memoire.txt" n'est pas lu et transféré dans
# la variable dans son intégralité
4 print(data)

```

Nous voyons ici que le script Python va bloquer jusqu'à ce que toutes les données de notre "mémoire" soient collectées sur le disque. Si vous exécutez cet exemple, vous verrez que 99% du temps d'exécution du programme est consacré à l'attente du collationnement sur le disque. Habituellement, ce n'est pas un problème pour les scripts simples comme celui-ci, cependant, dans d'autres situations, il aurait pu être possible d'accomplir d'autres tâches. Le cas typique est celui des requêtes effectuées sur les serveurs web. Imaginez que nous ayons un premier utilisateur qui lance une requête sur une base de données qui nécessite plus de 20 secondes avant d'envoyer la réponse. Si un second utilisateur adresse sur le même serveur une autre demande entre-temps, il devra attendre la clôture de la première réponse avant d'obtenir la sienne.

Pour résoudre ce problème, les serveurs Web, et principalement ceux orientés "Python" tels que Flask ou Django sont construits sur une interface Web particulière dite **WSGI** (Web Server Gateway Interface) dont les sous-processus sont capables de répondre de manière asynchrone aux requêtes.



Si l'un est occupé à traiter une longue requête, d'autres peuvent répondre aux nouvelles requêtes à venir.

Les E/S asynchrones ont été implantés dans Python 3.6 avec l'introduction, comme pour JS, des mots clés **async/await**. Tous les outils pour gérer ces E/S sont disponibles via la librairie `asyncio`. FastAPI s'appuie sur celle-ci, et c'est l'une des raisons pour lesquelles elle affiche d'aussi bonnes performances.

Voici un exemple très simple qui utilise les caractéristiques de cette librairie :

Exemple Basique en Python de `async` et `await`

```
1 import asyncio
2 async def affiche (message: str, nb_fois: int) -> None:
3     for i in range(nb_fois):
4         print(message)
5         await asyncio.sleep(1)
6
7 async def main():
8     await asyncio.gather(
9         affiche ("Bonjour les étudiant.e.s de ", 4),
10        affiche ("l'IED", 4),
11    )
12 asyncio.run(main())
```

Si nous souhaitons définir une fonction asynchrone, il suffit d'ajouter le mot-clé `async` avant le mot-clé "`def`". Cela nous permet d'utiliser le mot clé "`await`" à l'intérieur de la méthode. Ces fonctions asynchrones sont alors appelées **coroutines**.

A l'intérieur de notre méthode "`affiche`", dans la boucle, nous effectuons une première impression du "message", puis appelons la coroutine `asyncio.sleep`, qui est équivalente à `time.sleep(1)` et qui arrête le programme pendant un certain nombre de secondes (ici une). Notez que nous avons préfixé l'appel avec le mot clé "`await`" qui signifie que nous voulons attendre que cette coroutine se termine avant de continuer à boucler. C'est le principal avantage des mots-clés `async/await` : écrire du code qui ressemble à du code synchrone. Si nous avions omis l'attente, l'objet coroutine aurait été créé mais jamais exécuté. Enfin, notez que nous utilisons la



fonction **asyncio.run**. C'est elle qui créera une nouvelle boucle d'événements, exécutera notre coroutine et renverra son résultat. Cet exemple est certes pédagogique mais pas très intéressant d'un point de vue de l'asynchronicité ; puisque nous n'attendons qu'une seule opération, celui-ci n'est guère impressionnant.

3.4 Un Premier Exemple avec FastAPI

FastAPI, nous l'avons dit, est un framework qui se veut facile à utiliser et rapide à écrire. En fait, la création d'un point d'accès (endpoint) ne nécessite que quelques lignes de code :

Un Premier Exemple avec FastAPI

```
1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5
6 @app.get("/")
7 async def root():
8     return {"message": "Bonjour les Etudiant.e.s de la Licence Informatique"}
```

Dans cet exemple, nous avons défini un point d'accès "**GET**" au niveau de la racine ("`/`") et qui renvoie toujours la réponse JSON : `"message" : "Bonjour les Etudiant.e.s de la Licence Informatique"`. Pour ce faire, nous avons instancié un objet FastAPI attribué à `"app"`. *App* sera l'objet principal de l'application qui orientera toutes les routes de l'API. Nous avons ensuite créé une coroutine simple `async def root()` qui retourne ici un *dictionnaire* (clé, valeur) dont le message de retour est automatiquement gérée par FastAPI pour produire une réponse HTTP appropriée au format JSON à partir de ce dictionnaire.

La partie la plus importante du code est probablement la ligne commençant par `"@"`, et qui se trouve juste au-dessus de la définition de la coroutine et comporte un *décorateur*. En Python, un décorateur est *du sucre syntaxique* qui permet d'envelopper une fonction ou une classe sans compromettre la lisibilité. C'est à peu près équivalent à `app.get("/") (root)`. La plupart des frameworks comme Flask ou Django utilisent habilement ces décorateurs.



Le Sucre Syntaxique sur Wikipédia



FastAPI aura besoin que nous définissions un décorateur pour chaque méthode HTTP (GET, POST, PUT, ...) implémentée afin d'ajouter de nouveaux accès (ou URL) à notre application. Nous allons maintenant démarrer notre serveur "*Uvicorn*" et intégrer notre API.

Pour ce faire, le plus simple est de se créer un répertoire de travail (ici par exemple "essais" avec *mkdir*), de se rendre à la racine de celui-ci (*cd essais*) et d'enregistrer notre exemple ci-dessus avec le nom de votre choix (par ex "ma_premiere_api.py") dans notre répertoire.

Puis, dans le Terminal, de saisir la ligne de commandes suivante :

Lancement du Serveur Uvicorn

```
1 uvicorn ma_premiere_api:app --reload
```

L'option **-reload** nous permet de modifier à loisir notre programme "ma_premiere_api.py" sans que nous ayons besoin de relancer notre serveur. Si tout se passe pour le mieux, nous obtenons un affichage similaire à celui-ci :

Résultat du Lancement d'Uvicorn

```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press
        CTRL+C to quit)
INFO:     Started reloader process [26596] using watchdog
INFO:     Started server process [26598]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Ainsi, si nous ouvrons notre navigateur en saisissant l'URL indiquée (`http://localhost :8000`), nous voyons bien notre "message" s'afficher au format



JSON.

JSON affiché dans le navigateur

```
1 { "message": "Bonjour les Etudiant.e.s de la Licence  
Informatique" }
```

Et avec HTTPie, nous obtenons un résultat identique :

JSON affiché avec HTTPie et coloration syntaxique

```
1 http GET localhost:8000 # ligne de commandes saisie  
2  
3 HTTP/1.1 200 OK  
4 content-length: 65  
5 content-type: application/json  
6 date: Mon, 18 Jul 2022 21:20:26 GMT  
7 server: uvicorn  
8  
9 {  
10   "message": "Bonjour les Etudiant.e.s de la Licence  
11   Informatique"  
12 }
```

Comme nous l'avons mentionné dans la section sur les E/S asynchrones, FastAPI est une application compatible ASGI. Pour l'exécuter, nous avons besoin d'un serveur Web compatible avec ce protocole. Uvicorn constitue un très bon choix, parmi les serveurs existants : Gunicorn, Caddy 2, Gevent, Daphne, ... Dans le premier argument de la commande, il attend le nom du programme Python qui contient notre instance d'application, suivi de deux-points et du nom de la variable de notre application ASGI (dans notre exemple, il s'agit de l'application "app"). Il s'occupe ensuite de l'instancier et de la déployer sur notre machine locale.

L'une des fonctionnalités les plus appréciées de FastAPI est la création automatique de documentation interactive. Si nous saisissons l'adresse suivante (<http://localhost:8000/docs>) dans notre navigateur, nous devrions obtenir une interface Web qui ressemble à la figure 1 :

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI

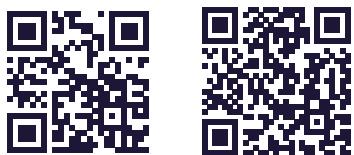


The screenshot shows the FastAPI interactive documentation for the 'default' endpoint. At the top, it says 'FastAPI 0.1.0 OAS3 /openapi.json'. Below that, under 'Parameters', it says 'No parameters'. Under 'Responses', there is a table with one row for status code 200, which is described as 'Successful Response'. The 'Media type' dropdown is set to 'application/json', with a note below it saying 'Controls Accept header.' and 'Example Value | Schema' showing the value 'string'.

- Fig. 1 - Documentation Interactive de FastAPI -

FastAPI répertoriera automatiquement tous les points d'accès et fournira une documentation détaillée sur les entrées et les sorties. Depuis cette interface graphique, nous pouvons même tester directement chaque méthode et accès. FastAPI s'appuie sur la spécification OpenAPI et sur les outils de [Swagger](#), en plus de Starlette et Pydantic.

Documentations de Starlette et Pydantic (pour en savoir plus)



3.5 Aller plus Loin avec FastAPI

Un tutoriel vidéo, voire deux valant mieux qu'un grand discours, je vous propose d'aller plus loin avec FastAPI en visionnant les deux tutoriaux suivants :



Apprendre et Transmettre sur YT)

Introduction à FastAPI :



Comment passer et récupérer des paramètres dans vos APIs :



Ceux-ci sont très complets et très bien structurés. Vous pouvez bien évidemment en trouver d'autres et dans la langue de votre choix. Ils vous seront utiles pour bien assimiler l'exemple de la section suivante.

3.6 Un Squelette d'Application à Adapter

Le plus simple pour bien comprendre le fonctionnement de FastAPI, ou de tout autre outil Python, consiste à analyser et tester un exemple illustratif complet. En voici un, emprunté à l'excellent site d'apprentissage "Real Python" :

Un Second Exemple plus Complet avec FastAPI

```
1 from fastapi import FastAPI, HTTPException
2 from mongita import MongitaClientDisk
3 from pydantic import BaseModel
4
5 class Artiste(BaseModel):
6     nom: str
7     album : str
8     nb_morceaux: int
9     id: int
10
11 app = FastAPI()
12
```

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



```
13 client = MongitaClientDisk()
14 db = client.db
15 Artistes = db.Artistes
16
17 @app.get("/")
18 async def root():
19     return {"message": "Bonjour et Bienvenue sur ma Base de Données Musicale"}
20
21 @app.get("/Artistes")
22 async def get_Artistes():
23     existing_Artistes = Artistes.find({})
24     return [
25         {key:Artiste[key] for key in Artiste if key != "_id"}
26         for Artiste in existing_Artistes
27     ]
28
29 @app.get("/Artistes/{Artiste_id}")
30 async def get_Artiste_by_id(Artiste_id: int):
31     if Artistes.count_documents({"id": Artiste_id}) > 0:
32         Artiste = Artistes.find_one({"id": Artiste_id})
33         return {key:Artiste[key] for key in Artiste if key != "_id"}
34     raise HTTPException(status_code=404, detail=f"Aucun artiste avec cet
35 identifiant {Artiste_id} trouvé")
36
37 @app.post("/Artistes")
38 async def post_Artiste(Artiste: Artiste):
39     Artistes.insert_one(Artiste.dict())
40     return Artiste
41
42 @app.put("/Artistes/{Artiste_id}")
43 async def update_Artiste(Artiste_id: int, Artiste: Artiste):
44     if Artistes.count_documents({"id": Artiste_id}) > 0:
45         Artistes.replace_one({"id": Artiste_id}, Artiste.dict())
46         return Artiste
47     raise HTTPException(status_code=404, detail=f"Aucun artiste avec cet
48 identifiant {Artiste_id} trouvé")
49
50 @app.put("/Artistes/upsert/{Artiste_id}")
51 async def update_Artiste(Artiste_id: int, Artiste: Artiste):
52     Artistes.replace_one({"id": Artiste_id}, Artiste.dict(), upsert=True)
53     return Artiste
54
55 @app.delete("/Artistes/{Artiste_id}")
56 async def delete_Artiste(Artiste_id: int):
57     delete_result = Artistes.delete_one({"id": Artiste_id})
58     if delete_result.deleted_count == 0:
```



```

57     raise HTTPException(status_code=404, detail=f"Aucun artiste avec cet
58     identifiant {Artiste_id} n'existe")
      return {"OK": True}

```

Nous voyons dans cet exemple que :

- Nous allons collecter, diffuser des données musicales structurées dans une classe "artiste" et qui comporte 4 éléments (le nom du groupe ou de l'artiste, le titre de l'album, le nombre de morceaux de l'album, l'identifiant de l'artiste). Notons que cette base est très maladroitement construite, car un artiste peut bien sûr sortir plusieurs albums... Il faudrait ici créer à minima deux tables distinctes ;
- La plupart des méthodes CRUD / REST sont implémentées : **GET, PUT, POST, DELETE** ;
- Nous avons utilisé une base de données de type NoSQL, dérivée de (Py)Mongo DB et appelée **Mongita**.

Avant d'exécuter ce code-source, il est donc nécessaire d'installer la **librairie Python Mongita** :

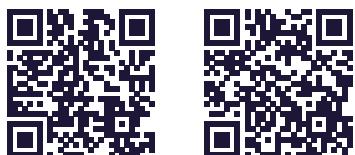
Installation de Mongita (2 méthodes)

```

pip install mongita
# ou encore si vous avez installé Anaconda Python (recommandé
)
conda install -c conda-forge mongita

```

Mongita sur PyPi et Github



Après avoir sauvegardé ce second exemple avec le nom de votre choix (ici 'mes_artistes.py') dans notre répertoire de travail (essais). Nous al-



lons maintenant redémarrer notre serveur "*Uvicorn*" et intégrer notre API. (**Crtl-C** permet d'arrêter le serveur précédemment lancé)

Puis, dans le Terminal, de saisir la ligne de commandes suivante :

Lancement du Serveur Uvicorn avec notre nouvelle base

```
1 uvicorn mes_artistes:app --reload
```

De même, si nous ouvrons notre navigateur en reprenant l'URL indiquée (<http://localhost:8000>), nous voyons bien notre nouveau "message" d'afficher au format JSON.

JSON affiché dans le navigateur

```
1 { "message": "Bonjour et Bienvenue sur ma Base de Données Musicale" }
```

Et aussi, en vérifiant avec **HTTPPie**, nous obtenons un résultat identique :

JSON affiché avec HTTPPie et coloration syntaxique

```
1
2 http GET localhost:8000      # ligne de commandes à saisir
3
4 HTTP/1.1 200 OK
5 content-length: 67
6 content-type: application/json
7 date: Tue, 19 Jul 2022 08:25:30 GMT
8 server: uvicorn
9
10 {
11     "message": "Bonjour et Bienvenue sur ma Base de Données
12     Musicale"
```

Notons que notre base de données ne comporte aucun artiste, elle est donc complètement vide. Plusieurs options se présentent à nous :

- Ajouter dans notre code, quelques lignes pour insérer directement nos



enregistrements (`insert_one`, `insert_many`) (voir la documentation de Mongita (cf QR-Code))

Ajouter des Enregistrements à Notre Base Musicale

```

1 # ....
2
3 client = MongitaClientDisk()
4 db = client.db
5 Artistes = db.Artistes
6
7 # Insertion de nos artistes (ici groupes) et albums
8
9 Artistes.insert_one ({ "nom": "Delain", "album": "Apocalypse And Chill", "
10      nb_morceaux": 8, "id" : 1})
11 Artistes.insert_one ({ "nom": "Nightwish", "album": "Decades", "nb_morceaux": 22,
12      "id" : 2})
13 Artistes.insert_one ({ "nom": "Epica", "album": "Omega", "nb_morceaux": 12, "
14      id" : 3})
15 Artistes.insert_one ({ "nom": "Leaves'Eyes", "album": "The Last Viking", "
16      nb_morceaux": 28, "id" : 4})
17 Artistes.insert_one ({ "nom": "Avantasia", "album": "The Metal Opera Part II",
18      "nb_morceaux": 10, "id" : 5})
19
20 # ....

```

Il est possible également d'utiliser la méthode "`insert_many()`" qui prend une liste en argument :

Ajouter une Liste d'Enregistrements à Notre Base Musicale

```

1 # ....
2
3 client = MongitaClientDisk()
4 db = client.db
5 Artistes = db.Artistes
6
7 # Insertion de nos artistes (ici groupes) et albums sous la forme d'une liste
8
9 Artistes.insert_many([
10      {"nom": "Delain", "album": "Apocalypse And Chill", "nb_morceaux": 8,
11      "id" : 1},
12      {"nom": "Nightwish", "album": "Decades", "nb_morceaux": 22, "id" :
13      2}
14 ]
)

```



```

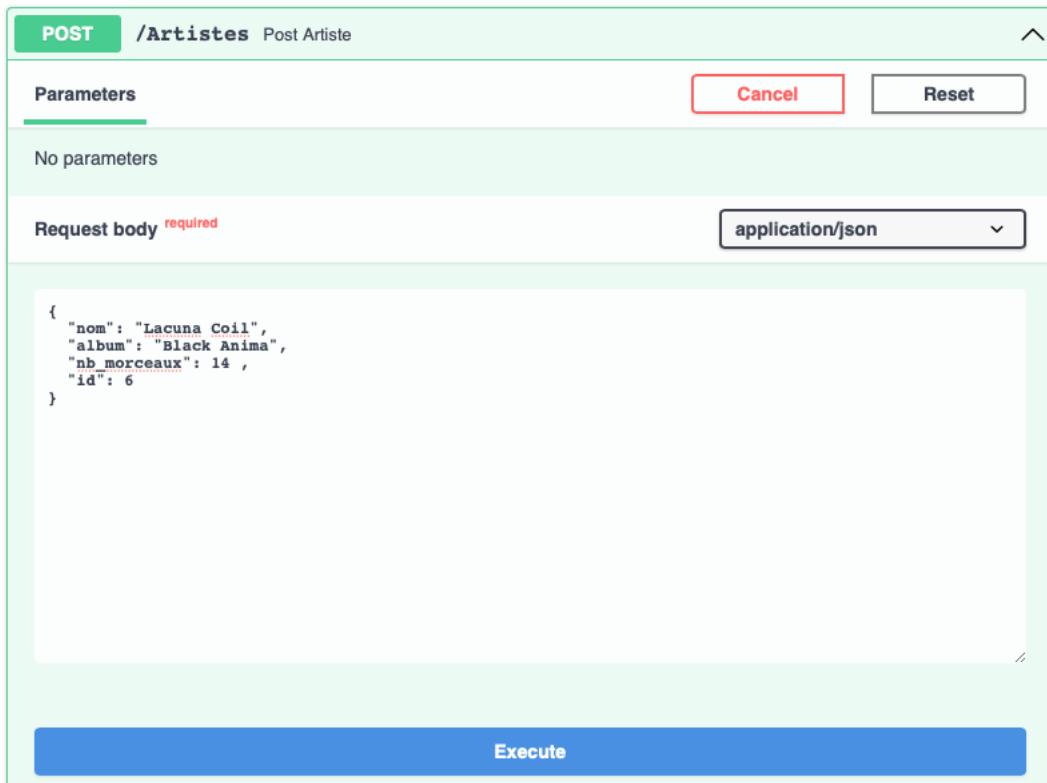
12     2},
13         {"nom": "Epica", "album": "Omega", "nb_morceaux": 12, "id": 3},
14         {"nom": "Leaves'Eyes", "album": "The Last Viking", "nb_morceaux":
15             28, "id": 4},
16             {"nom": "Avantasia", "album": "The Metal Opera Part II ", "
17                 nb_morceaux": 10, "id": 5}
18             ])
19
20
21 # .../...

```

- D'importer en local ou depuis le web un fichier JSON existant, d'adapter des données existantes, etc (voir la documentation de Mongo DB et PyMongo par exemple).
- D'utiliser HTTPie ou encore l'interface de documentation (<http://localhost:8000/docs>) pour insérer des données. Nous allons nous rendre sur cette dernière :

Après avoir ajouté nos cinq artistes et albums (l'**option -reload** nous permet de modifier notre fichier sans avoir à relancer notre serveur), rendons-nous sur la page de la documentation (interface graphique) en ajoutant "**/docs**" à notre URL. Sélectionnons "**POST**" puis "**Try it out**"... Nous pouvons directement saisir notre sixième groupe et album comme le montre la figure 2.

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



- Fig. 2 - Saisie d'un Enregistrement depuis la Documentation -

Quand la saisie est terminée, il suffit de cliquer sur "**Execute**". Notre artiste et ses données sont bien importés (Code 200, code conforme vu au chapitre 2) (fig.3).

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/Artistes' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "nom": "Lacuna Coil",
    "album": "Black Anima",
    "nb_morceaux": 14 ,
    "id": 6
}'
```



Request URL

```
http://localhost:8000/Artistes
```

Server response

Code	Details
------	---------

200

Response body

```
{
  "nom": "Lacuna Coil",
  "album": "Black Anima",
  "nb_morceaux": 14,
  "id": 6
}
```



Download

Response headers

```
content-length: 67
content-type: application/json
date: Tue,19 Jul 2022 09:24:53 GMT
server: uvicorn
```

Responses

Code	Description	Links
------	-------------	-------

200

Successful Response

No links

- Fig. 3 - Validation des Données Saisies depuis la Documentation -

Nous pouvons grâce à l'item **GET (Get Artistes)** ou dans **GET (Get Artiste By id)** vérifier que notre enregistrement a bien complété notre base existante. Une autre possibilité est d'utiliser HTTPPie :

Vérification avec HTTPPie

```
1
2 http GET http://localhost:8000/Artistes/6    # ligne de
      commandes à saisir
3
4 HTTP/1.1 200 OK
```



```

5 content-length: 67
6 content-type: application/json
7 date: Tue, 19 Jul 2022 09:30:23 GMT
8 server: uvicorn
9
10 {
11     "album": "Black Anima",
12     "id": 6,
13     "nb_morceaux": 14,
14     "nom": "Lacuna Coil"
15 }
```

Ces valeurs étant purement indicatives, vous pouvez bien évidemment adapter cette base musicale avec vos goûts personnels, en ajoutant, modifiant ou supprimant les enregistrements de votre choix.

3.7 Les Exercices du TP

Pour ce troisième chapitre, l'**exercice 3.1** est assez facile à réaliser. L'**exercice 3.2** qui explore la reconnaissance faciale est pour un Bonus bien mérité.

Exercice 3.1 (A Rendre)

En vous inspirant du code-source contenu dans la section "Squelette d'Application à Adapter" :

 **Développez votre propre application FastAPI** dans la thématique de votre choix, en y insérant un jeu de données à consulter et à partager collaborativement.

Cela pourrait-être (par exemple) :

- 1) Votre playlist (YouTube, Spotify, Deezer, Apple Music, ...)
- 2) Des données issues de l'Open Data (data.gouv.fr) comme celles de la santé, de la culture, des sports, ... ou autre (voir la liste des open API du chapitre 2)
- 3) Des articles sélectionnés et collectés sur des blogs ou sur des réseaux sociaux ([Twitter](#), [Meta \(Facebook\)](#), [Instagram \(Instagrapgi\)](#), des vidéos [TikTok](#), [Snapchat](#),...)



- 4) Une base de données personnelle que vous souhaitez partager (musique, peinture, BD, livres que vous avez adorés, lieux à visiter, restaurants à ne pas manquer, bars à cocktail, catalogue d'une collection d'objets,...).
 - ✓ Essayez de limiter votre jeu de données entre 20 (minimum) et 100 (maximum) enregistrements.
 - ✓ Comme à l'habitude, n'oubliez pas de commenter vos sources, d'expliquer les traitements opérés sur les données, d'illustrer votre réponse par de nombreuses copies d'écran (HTTPie, interface "/docs", Terminal, ...) et d'ajouter vos fichiers dans l'archive attachée.

Exercice 3.2 (Bonus)

Après avoir regardé les "Compléments pour le Bonus" ci-après et téléchargé l'archive "Bonus 3.2.zip" sur Moodle :

⌚ Réalisez une "**Solution de Reconnaissance**" de votre choix en adaptant et modifiant les fichiers joints :

- 1) Avec un flux vidéo, comme dans l'exemple, en utilisant la webcam
 - 2) Avec un jeu d'images sélectionnées et enregistrées en local.
- ✓ Pour vous aider, sur [Github](#) et dans l'article sur le site (cf QR-Code) [Kongagura](#) "Haar cascade - Détection de visages et d'objets" vous trouverez les fichiers XML pour la détection de sourires, d'yeux, de lunettes, de tête de chat, de plaques d'immatriculation, et en cherchant un peu sur Google : de [chiens](#), de célébrité, etc....
- ✓ Proposez de trois à cinq exemples maximum, idéalement avec une image qui ne fonctionne pas (cf Petit Guide)
- ✓ Les Audacieux.cieuses pourront choisir la librairie et le classifieur de leur choix (scikit learn, tensorflow, pytorch, keras, ...)
- ✓ Pensez à bien expliquer vos choix et méthodes et à présenter les outils employés. Faites les C.E(s) illustratives habituelles sans oublier d'ajouter les fichiers l'archive jointe.



3.7.1 Compléments pour le Bonus : Implémenter une Fonction de Reconnaissance de Visages en Temps Réel

Pour réaliser l'exercice Bonus, nous aurons besoin d'implanter un système de détection de visages qui fonctionne en temps réel avec une entrée vidéo, en l'occurrence ici **la webcam de notre ordinateur**. Outre les accès que nous avons vus tout au long de ce chapitre, FastAPI a également la capacité de gérer des points d'E/S de type **WebSocket**, ce qui nous permet d'envoyer et de recevoir des flux de données. Pour effectuer cela, le navigateur enverra dans la WebSocket un flux d'images issu de notre webcam intégré, et notre application FastAPI exécutera l'algorithme de reconnaissance faciale et renverra les coordonnées du ou des visages détectés dans l'image sous la forme d'un ou plusieurs **rectangles colorés en vert**. Pour que cette tâche de détection fonctionne correctement, nous aurons besoin de la librairie Python **OpenCV**, de codes Python, HTML et Javascript adaptés et tirés du livre et du [Github](#) de François Voron à télécharger dans l'espace "Outils Informatiques Collaboratifs" sur Moodle, et de lancer en local deux serveurs : **uvicorn** et le serveur embarqué **http.server** de Python.

Installation d'OpenCV

La vision par ordinateur est un domaine lié à l'apprentissage automatique qui vise à développer des algorithmes et des systèmes pour analyser automatiquement des images et des vidéos. Un exemple typique d'application de la vision par ordinateur (computer vision) est la reconnaissance faciale : un système détectant automatiquement les visages humains dans une image. Pour nous aider dans cette tâche, nous allons utiliser *OpenCV*, qui est l'une des bibliothèques de computer vision la plus simple à utiliser (*scikit-learn* aurait pu convenir également, mais reste un peu plus complexe pour notre TP). Cette librairie est écrite en C et C++ et peut être utilisée dans de nombreux langages de programmation.

Nous avons besoin bien évidemment d'une *webcam* intégrée ou connectée à votre port USB et d'installer la librairie *OpenCV* qui s'opère de manière habituelle :

Installation d'OpenCV

```
pip install opencv-python  
# ou selon
```



```
pip3 install opencv-python
```

A la date du cours, il s'agit de la version 4.6.0 (ou supérieure) qui sera installée sur notre ordinateur.

Un Premier Programme de Computer Vision à Tester en Local

Il est important de tester le bon fonctionnement d'OpenCV en saisissant puis en lançant dans le Terminal le programme suivant (inclus dans l'archive "Bonus 3.2.zip" à télécharger dans l'espace de cours) :

Test d'OpenCV

```

1 import cv2
2
3 # Chargement du modèle d'apprentissage
4 face_cascade = cv2.CascadeClassifier(
5     cv2.data.haarcascades + "haarcascade_frontalface_default .xml"
6 )
7
8 # En fonction de votre configuration changer la valeur 0 en 1 ou 2 ...
9 video_capture = cv2.VideoCapture(0)
10
11 while True:
12     # récupération du flux d'images
13     ret, frame = video_capture.read()
14
15     # conversion en niveaux de gris et détection
16     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
17     faces = face_cascade.detectMultiScale(gray)
18
19     # dessin d'un rectangle autour du ou des visages reconnus
20     for (x, y, w, h) in faces:
21         cv2.rectangle(
22             img=frame,
23             pt1=(x, y),
24             pt2=(x + w, y + h),
25             color=(0, 255, 0),
26             thickness=2,
27         )
28

```



```

29
30 # Affichage du résultat
31 cv2.imshow("Test OpenCV", frame)
32
33 # si la touche "q" est pressée -> arrêt
34 if cv2.waitKey(1) == ord("q"):
35     break
36
37 video_capture.release()
38 cv2.destroyAllWindows()

```

Attention

Plusieurs erreurs peuvent être générées par OpenCV :

- OpenCV : AVFoundation didn't find any attached Video Input Devices!
- OpenCV : camera failed to properly initialize!

Il faut dans ce cas **changer la valeur d'index** :

video_capture = cv2.VideoCapture(1)

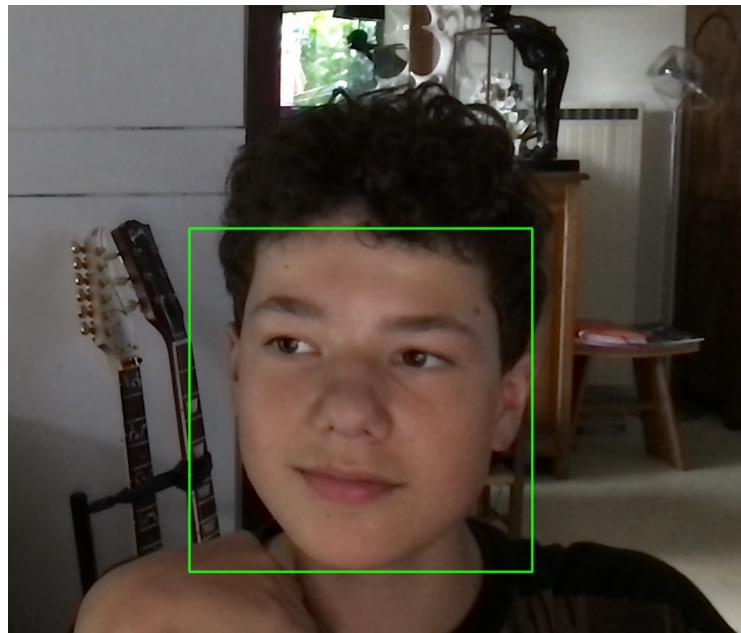
ou

video_capture = cv2.VideoCapture(2)

- OpenCV : not authorized to capture video (status 0), requesting... Terminated due to signal : ABORT TRAP (6)

Cette dernière erreur est **récurrente sur MacOS**. Il faut autoriser le **Terminal** à utiliser **l'Appareil Photo**, dans l'onglet Sécurité et Confidentialité et en saisissant le mot de passe administrateur.

En exécutant le programme ci-dessous dans le Terminal : **python test_opencv.py**, nous obtenons (Fig.4) :



- Fig. 4 - Détection du Visage Réussie -

Lorsque l'algorithme détecte un visage, il dessine un rectangle vert autour de celui-ci, comme sur la figure 4. Appuyons sur la touche "q" de notre clavier pour arrêter le script comme indiqué dans le code.

Expliquons en quelques lignes le code ci-dessous. Nous instancions la classe *CascadeClassifier* avec un fichier XML qui est fourni avec la bibliothèque *OpenCV*. Cette classe est en fait un algorithme d'apprentissage automatique utilisant la *méthodes des cascades* (ou ondelettes) de *Haar* (voir le QR-Code pour en savoir plus). L'avantage d'*OpenCV* est que cette librairie est livrée avec des modèles pré-formatés, fournis sous la forme de fichiers XML, comme celui pour la détection de visages que nous utilisons ici.

Nous instancions ensuite une classe *VideoCapture* qui nous permet de diffuser des images à partir d'une webcam. L'argument entier dans l'initialiseur est l'index (à changer au besoin) de la caméra que nous souhaitons utiliser. Nous voyons ensuite une boucle infinie afin de pouvoir exécuter en continu des détections sur le flux d'images ; nous récupérons une première image (frame) depuis l'instance *video_capture*, qui est ensuite envoyée au classifieur grâce à la méthode *detectMultiScale* et ainsi de suite... Notez que nous l'avons d'abord convertie en niveaux de gris, ce qui constitue une exigence pour les classificateurs qui utilisent *les cascades de Haar*.



Le résultat de cette opération est une liste de "tuples" contenant les caractéristiques *des rectangles autour des visages détectés* : x et y sont les coordonnées du point de départ ; w et h sont la largeur et la hauteur du ou des rectangles. Génial, il nous a fallu moins de 30 lignes de code pour obtenir un système de reconnaissance de visage fonctionnel.

Les Cascades de Haar sur Kongakura et OpenCV



Implémentation d'OpenCV avec FastAPI

Commençons par présenter le code-source que nous allons tester à la fois avec HTTPie et l'interface graphique "/docs" :

Code-Source de "detection.py" (inclus)

```

1  from typing import List, Tuple
2
3
4  import cv2
5  import numpy as np
6  from fastapi import FastAPI, File, UploadFile
7  from pydantic import BaseModel
8
9
10 app = FastAPI()
11 cascade_classifier = cv2.CascadeClassifier()
12
13
14 class Faces(BaseModel):
15     faces: List[Tuple[int, int, int, int]]
16
17
18 @app.post("/face-detection", response_model=Faces)
19 async def face_detection(image: UploadFile = File(...)) -> Faces:
20     data = np.fromfile(image.file, dtype=np.uint8)
21     image = cv2.imdecode(data, cv2.IMREAD_UNCHANGED)
22     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

```



```

23     faces = cascade_classifier . detectMultiScale(gray)
24     if len(faces) > 0:
25         faces_output = Faces(faces=faces.tolist () )
26     else:
27         faces_output = Faces(faces=[])
28     return faces_output
29
30
31 @app.on_event( "startup" )
32 async def startup():
33     cascade_classifier . load(
34         cv2.data.haarcascades + "haarcascade_frontalface_default . xml"
35     )

```

Comme vous pouvez le voir, cette application FastAPI reste relativement assez simple. En haut du code, nous instancions une classe **CascadeClassifier** et nous définissons un point d'accès "**face_detection**" (entry point) qui attend que nous passions en paramètres une ou plusieurs images via "**UploadFile**". En effet, si nous ne spécifions pas d'image(s), notre application n'est pas fonctionnelle et retournera un message JSON d'erreur. Une fois que nous avons récupéré notre (ou nos) image(s), nous effectuons deux opérations en utilisant les librairies **NumPy** et OpenCV car les images doivent être chargées dans une **matrice NumPy** utilisable par OpenCV. En utilisant NumPy, nous pouvons charger les données binaires dans un tableau de pixels (**data**). Tableau qui sera utilisé par la suite par la fonction `imdecode` pour créer une matrice OpenCV appropriée. Enfin, nous pouvons exécuter la prédiction à l'aide du classificateur. Notez que nous structurons le résultat dans un modèle **Pydantic** sous la forme d'un **Tuple de 4 entiers** pour définir notre rectangle. Lorsque notre classifieur détecte des visages, il renvoie le résultat sous la forme d'un **tableau NumPy** imbriqué, transformé en **liste** par la méthode `tolist()`.

Nous pouvons maintenant lancer notre serveur uvicorn :

Lancement du Serveur Unicorn

```
1 uvicorn detection:app --reload
```

Si maintenant, dans une autre fenêtre du Terminal, nous lançons la ligne



de commandes HTTPie :

Lignes de Commandes HTTPie

```

1 http GET http://localhost:8000/ # puis entrée
2
3
4
5 HTTP/1.1 404 Not Found
6 content-length: 22
7 content-type: application/json
8 date: Wed, 20 Jul 2022 10:27:23 GMT
9 server: uvicorn
10
11 {
12     "detail": "Not Found"
13 }
14
15 http GET http://localhost:8000/face-detection # puis entrée
16
17 HTTP/1.1 405 Method Not Allowed
18 content-length: 31
19 content-type: application/json
20 date: Wed, 20 Jul 2022 10:32:52 GMT
21 server: uvicorn
22
23 {
24     "detail": "Method Not Allowed"
25 }
```

Nous voyons effectivement un message d'erreur selon les points d'accès : "**detail Not Found**" ou "**Method Not Allowed**" car nous n'avons pas spécifié d'image(s) à notre application. Nous allons remédier à cela en fournissant à notre application une image (inclus dans l'archive) en paramètre :

Lignes de Commandes HTTPie avec Paramètre

```

1
2 http --form POST http://localhost:8000/face-detection
3     image@students.jpg # puis entrée
4
5 HTTP/1.1 200 OK
6 content-length: 43
```



```
6 content-type: application/json
7 date: Wed, 20 Jul 2022 10:38:38 GMT
8 server: uvicorn
9
10 {
11     "faces": [
12         [
13             237,
14             92,
15             80,
16             80
17         ],
18         [
19             426,
20             75,
21             115,
22             115
23         ]
24     ]
25 }
```

Nous utilisons ici la méthode **POST** pour envoyer notre image "**students.jpg**", l'**option –form** permettant que l'image soit encodée correctement. Ainsi pour notre image, deux visages complets ont été détectés.

Utilisons maintenant notre interface graphique en saisissant dans notre navigateur l'adresse suivante (<http://localhost:8000/docs>). Il nous suffit de cliquer sur le bouton **POST /face-detection** (nous voyons que le champ "image" est requis) puis "**Try It Out**" pour pouvoir sélectionner l'image de notre choix et enfin "**Execute**" pour lancer le classifieur.

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



The screenshot shows a FastAPI endpoint for face detection. At the top, there is a form field labeled "image * required" with the type "string(\$binary)" and a "Choisir le fichier" button containing the file "students.jpg". Below the form are two buttons: "Execute" and "Clear". The "Responses" section contains a "Curl" command and a "Request URL". The "Server response" section shows a successful 200 status code with a JSON response body:

```
{ "faces": [ [ 237, 92, 80, 80 ], [ 426, 75, 115, 115 ] ] }
```

At the bottom right of the response area are "Copy" and "Download" buttons.

- Fig. 5 - Sélection d'une Image et Exécution -

A tester...

Combien de visages sont détectés par notre classifieur et notre application FastAPI avec les fichiers inclus (student2.jpg, student3.jpg, student4.jpg)?

Attention

Si ces lignes de commandes HTTPie de passage de paramètres vous renvoient quelques erreurs, ajoutez la librairie "multipart" par la commande : pip install python-multipart.



Utilisation (Facultative) des WebSockets avec FastAPI

L'un des principaux avantages des WebSockets est qu'elles permettent l'ouverture d'un canal de communication en duplex intégral entre le client et le serveur. Une fois la connexion établie, les messages peuvent être transmis rapidement sans avoir à passer par toutes les étapes du protocole HTTP. Par conséquent, elles sont beaucoup plus adaptées à l'envoi de nombreux messages en temps réel.

Nous allons donc implémenter une WebSocket qui est capable à la fois d'accepter le flux d'images et d'exécuter la détection de visages d'OpenCV. Comme nous recevrons certainement plus d'images du navigateur que le serveur ne peut en gérer, en raison du temps nécessaire pour exécuter l'algorithme de détection, nous devrons travailler avec une file d'attente (ou tampon) de taille limitée et déposer quelques images pour gérer le flux en temps (quasi) réel.

Code-Source de "detection2.py" (inclus)

```

1 import asyncio
2 from typing import List, Tuple
3
4 import cv2
5 import numpy as np
6 from fastapi import FastAPI, WebSocket, WebSocketDisconnect
7 from pydantic import BaseModel
8
9
10 app = FastAPI()
11 cascade_classifier = cv2.CascadeClassifier()
12
13
14 class Faces(BaseModel):
15     faces: List[Tuple[int, int, int, int]]
16
17
18 async def receive(websocket: WebSocket, queue: asyncio.Queue):
19     bytes = await websocket.receive_bytes()
20     try:
21         queue.put_nowait(bytes)
22     except asyncio.QueueFull:
23         pass
24
25

```

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



```
26  async def detect(websocket: WebSocket, queue: asyncio.Queue):
27      while True:
28          bytes = await queue.get()
29          data = np.frombuffer(bytes, dtype=np.uint8)
30          img = cv2.imdecode(data, 1)
31          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
32          faces = cascade_classifier .detectMultiScale(gray)
33          if len(faces) > 0:
34              faces_output = Faces(faces=faces.tolist () )
35          else:
36              faces_output = Faces(faces=[])
37          await websocket.send_json(faces_output.dict())
38
39
40 @app.websocket("/face-detection")
41 async def face_detection(websocket: WebSocket):
42     await websocket.accept()
43     queue: asyncio.Queue = asyncio.Queue(maxsize=10)
44     detect_task = asyncio.create_task(detect(websocket, queue))
45     try:
46         while True:
47             await receive(websocket, queue)
48     except WebSocketDisconnect:
49         detect_task.cancel()
50         await websocket.close()
51
52
53 @app.on_event("startup")
54 async def startup():
55     cascade_classifier .load(
56         cv2.data.haarcascades + "haarcascade_frontalface_default .xml"
57     )
```

Comme nous le voyons dans le code ci-dessus, nous avons utilisé deux sous-routines principales : **receive** et **detect**. La première sert à lire les octets issus de la **WebSocket**, tandis que la seconde sert à effectuer la détection et à renvoyer le résultat. L'objet **asyncio.Queue** nous permet de mettre dans une file d'attente une collection limitée d'images en mémoire (ici 10) et de les récupérer selon la stratégie premier entré, premier sorti (**FIFO**).

La coroutine "**receive**" reçoit des données et les place à la fin de la file d'attente. Lorsque nous travaillons avec "**queue**", nous avons à notre disposition deux méthodes pour placer un nouvel élément dans la file d'attente :



put et put_nowait. Sans trop rentrer dans les détails, nous utilisons ici *put_nowait...*

Nous ne souhaitons pas que ces deux sous-routines soient en concurrences : nous voulons accepter de nouvelles images et exécuter en continu des détections sur les images au fur et à mesure qu'elles arrivent dans le flux vidéo. C'est pourquoi la fonction de détection a sa propre boucle infinie. En utilisant `asyncio.create_task`, nous la planifions dans la boucle d'événements afin qu'elle commence à gérer les images de la file d'attente. Nous voyons ensuite la boucle WebSocket habituelle, qui appelle la méthode (*await*) `receive()`. D'une certaine manière, nous pourrions dire que cette détection s'exécute "**en arrière-plan**". Notez que le programme vérifie que cette tâche est annulée lorsque la WebSocket est fermée afin que la boucle infinie soit correctement stoppée.

Envoi (Facultatif) d'un Flux d'images depuis le Navigateur

Sans trop rentrer dans les détails, nous allons capturer des images de la webcam directement dans le navigateur et nous allons les renvoyer à notre application par l'intermédiaire d'une WebSocket. Pour ce faire, nous allons utiliser le code JavaScript ci-dessous (inclus dans l'archive).

Les différentes étapes consistent grossièrement à activer la webcam dans le navigateur (grâce à l'API du navigateur "**mediaDevices**", nous pouvons répertorier toutes les webcams disponibles) et afficher un formulaire à l'utilisateur pour qu'il sélectionne celle de son choix (index.html, inclus)), à ouvrir la connexion de la WebSocket, à sélectionner une image de caméra et à l'envoyer par l'intermédiaire de la WebSocket pour que l'algorithme de reconnaissance de visages trace un rectangle autour de celui-ci.

Le Code-Source de script.js (Inclus)

```

1 const IMAGE_INTERVAL_MS = 42;
2 // pour 24 images par seconde
3
4 const drawFaceRectangles = (video, canvas, faces) => {
5   const ctx = canvas.getContext('2d');
6
7   ctx.width = video.videoWidth;
8   ctx.height = video.videoHeight;
9

```

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



```
10    ctx.beginPath();
11    ctx.clearRect(0, 0, ctx.width, ctx.height);
12    for (const [x, y, width, height] of faces.faces) {
13        ctx.strokeStyle = "#49fb35";
14        ctx.beginPath();
15        ctx.rect(x, y, width, height);
16        ctx.stroke();
17    }
18 };
19
20 const startFaceDetection = (video, canvas, deviceId) => {
21     // point d'accès de notre serveur uvicorn et application
22
23     const socket = new WebSocket('ws://localhost:8000/
24         face-detection');
24     let intervalId;
25
26     // La connexion est ouverte
27     socket.addEventListener('open', function () {
28
29         // Lecture de la vidéo depuis la webcam
30         navigator.mediaDevices.getUserMedia({
31             audio: false,
32             video: {
33                 deviceId,
34                 width: { max: 640 },
35                 height: { max: 480 },
36             },
37         }).then(function (stream) {
38             video.srcObject = stream;
39             video.play().then(() => {
40                 // Adaptation des dimensions canvas <-> vidéo
41                 canvas.width = video.videoWidth;
42                 canvas.height = video.videoHeight;
43
44                 // Envoi d'une image dans la socket Web toutes les
45                 ms (soit 24 images par seconde)
46                 intervalId = setInterval(() => {
47
48                     // Création d'un canvas virtuel pour l'image vidéo
49                     // capturée
50                     const canvas = document.createElement('canvas');
51                     const ctx = canvas.getContext('2d');
52                     canvas.width = video.videoWidth;
53                     canvas.height = video.videoHeight;
54                     ctx.drawImage(video, 0, 0);
```

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



```
53         // Conversion en JPEG et envoi dans la WebSocket
54         canvas.toBlob((blob) => socket.send(blob), 'image
55             /jpeg');
56             }, IMAGE_INTERVAL_MS);
57         });
58     });
59 );
60
61 // En écoute d'un message
62 socket.addEventListener('message', function (event) {
63     drawFaceRectangles(video, canvas, JSON.parse(event.data
64     ));
65 });
66
67 // On ferme tout
68 socket.addEventListener('close', function () {
69     window.clearInterval(intervalId);
70     video.pause();
71 });
72
73     return socket;
74 };
75
76 window.addEventListener('DOMContentLoaded', (event) => {
77     const video = document.getElementById('video');
78     const canvas = document.getElementById('canvas');
79     const cameraSelect = document.getElementById(
80         'camera-select');
81     let socket;
82
83     // Recherche des webcams disponibles
84     navigator.mediaDevices.enumerateDevices().then((devices)
85         => {
86         for (const device of devices) {
87             if (device.kind === 'videoinput' && device.deviceId)
88             {
89                 const deviceOption = document.createElement('option
90                     ');
91                 deviceOption.value = device.deviceId;
92                 deviceOption.innerText = device.label;
93                 cameraSelect.appendChild(deviceOption);
94             }
95         }
96     });
97 }
```

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



```
93 // Si 'submit', commencement de la reconnaissance de  
94 visages  
95 document.getElementById('form-connect').addEventListener  
96 ('submit', (event) => {  
97   event.preventDefault();  
98  
99   // Fermeture de la socket précédente si c'est le cas  
100  if (socket) {  
101    socket.close();  
102  }  
103  
104  const deviceId = cameraSelect.selectedOptions[0].value;  
105  socket = startFaceDetection(video, canvas, deviceId);  
106});  
107});
```

Le Code-Source de la Page HTML et du Formulaire (Inclus)

```
1 <!doctype html>  
2 <html lang="fr">  
3  
4 <head>  
5   <meta charset="utf-8">  
6   <meta name="viewport" content="width=device-width,  
7     initial-scale=1">  
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/  
9     dist/css/bootstrap.min.css" rel="stylesheet" integrity="  
10    sha384-EVSTQN3/  
11    azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpueC0mLASjC  
12    " crossorigin="anonymous">  
13   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5  
14     .0.2/dist/js/bootstrap.bundle.min.js" integrity="  
15    sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP  
16    +JcXn/tWtIaxVXM" crossorigin="anonymous"></script>  
17  
18   <title>Detection de Visages</title>  
19 </head>  
20  
21 <body>  
22   <div class="container">  
23     <h1 class="my-3">Detection de Visages</h1>  
24     <form id="form-connect">
```



```

17   <div class="input-group mb-3">
18     <select id="camera-select"></select>
19     <button class="btn btn-success" type="submit" id="button-start">Demarrer</button>
20   </div>
21 </form>
22 <div class="position-relative">
23   <video id="video"></video>
24   <canvas id="canvas" class="position-absolute top-0
start-0"></canvas>
25 </div>
26 </div>
27
28   <script src="script.js"></script>
29 </body>
30
31 </html>

```

Exécution (Facultative) de nos Programmes Python en local sur Deux Serveurs

Notre système de reconnaissance de visages est maintenant prêt et il est temps de l'essayer. Nous allons donc démarrer deux serveurs : **Uvicorn** pour notre application FastAPI (**detection.py**) et le **serveur Python** intégré pour le fichier HTML et le script JS.

Après avoir dézippé tous les fichiers de "**Bonus 3.2.zip**" dans le répertoire de votre choix (ici nous avons gardé "essais", dans une première fenêtre de terminal, lançons notre application FastAPI "detection.py" :

Lancement du Serveur Uvicorn

```
1 uvicorn detection:app --reload
```

Dans une seconde fenêtre de terminal, dans le même répertoire "essais" (important), lançons le serveur intégré de Python avec notre fichier (index.html) avec un numéro de port différent (ici 9000, cf cours L2-RES) :



Lancement du Serveur Python

```
1 python -m http.server --directory websocket 9000
```

L'application est maintenant lancée sur le port 9000. Nous pouvons y accéder dans votre navigateur avec l'adresse (<http://localhost:9000>). Vous verrez une page HTML très simple vous invitant à choisir la caméra que vous souhaitez utiliser, puis cliquez sur le bouton "Demarrer" pour lancer la reconnaissance...

Attention

En fonction du système, notamment sur **MacOS**, ou du choix du navigateur, bien que la page s'affiche correctement, **la détection de la webcam** ne fonctionne pas ou quelquefois c'est la **liaison websocket** est qui inopérante. Si ces erreurs sont détectées, en attendant que je trouve une solution viable, le plus simple est de ne pas tenter pour l'instant les sections "**facultatives**".

3.8 Quelques Articles et Livres pour Aller plus Loin...

Quelques Sites et Ouvrages à Explorer...

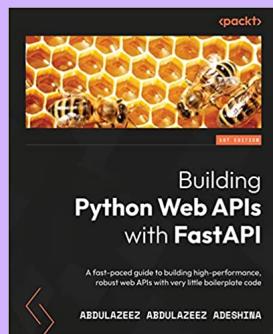


- (1) Le livre de François Voron dans lequel nous nous sommes très largement inspirés pour les activités et les code-sources chez Packt Pub, "Building Data Science Applications with FastAPI : Develop, manage, and deploy efficient machine learning applications with Python", oct. 2021 :

CHAPITRE 3. DÉVELOPPER UNE RESTFUL API AVEC FASTAPI



- (2) Toujours chez Packt Pub, "Building Web APIs with FastAPI and Python : A fast-paced guide to building high-performance, robust web APIs with very little boilerplate code", Août 2022 de Abdulazeez Abdulazeez Adeshina :



!

- (3) Une Introduction en 3 parties à FastAPI sur Dev.to par Eric le Codeur :



- (4) Un Rappel du Didacticiel FastAPI en langue française sur Linux Tut (et suivantes) :



- (5) Un très très intéressant article sur Real Python (comme beaucoup sur ce site) "Using FastAPI to Build Python Web APIs"



avec lequel nous nous sommes inspirés pour ce chapitre.



- (6) Quelques articles à lire sur le site "LeDataScientist" pour les passionné.e.s de ML, de programmation python et pour les futur.e.s étudiant.e.s optant pour le master Big Data.



Quatrième partie

**TP 4 - Médias Immersifs, Réalité
Virtuelle et Partages
Collaboratifs**

Chapitre 4

Mondes Virtuels Immersifs et Plateforme Collaborative

ÀVEC le développement des technologies dites immersives, les possibilités de créer des mondes virtuels détaillés dans lesquels les utilisateurs peuvent collectivement s'immerger sont de plus en plus nombreuses. L'immersion dont il est question ici désigne l'effet par lequel les utilisateurs perçoivent l'environnement virtuel comme très proche de la réalité. Ce terme est également souvent utilisé en relation avec la réalité virtuelle (VR) ou Virtual Reality. La VR, en particulier, permet un degré d'immersion très élevé. Même s'il ne s'agit que d'un environnement généré par ordinateur, l'expérience visuelle donne à notre cerveau et à nos sens l'impression que le monde virtuel est réel.

La VR est très utile dans de nombreux domaines : elle peut être utilisée pour dispenser des cours en ligne, pour des conférences d'affaires qui pourront être améliorées grâce à un haut degré d'immersion, pour des applications médicales, par exemple, pour préparer des procédures chirurgicales complexes... Dans une première partie du TP, nous verrons l'exemple (ancien) de Second Life puis, en attendant que ces technologies deviennent aisément accessibles, nous utiliserons le framework Streamlit pour partager facilement en ligne des données qu'elles soient issues de la Data Science ou des données personnelles.



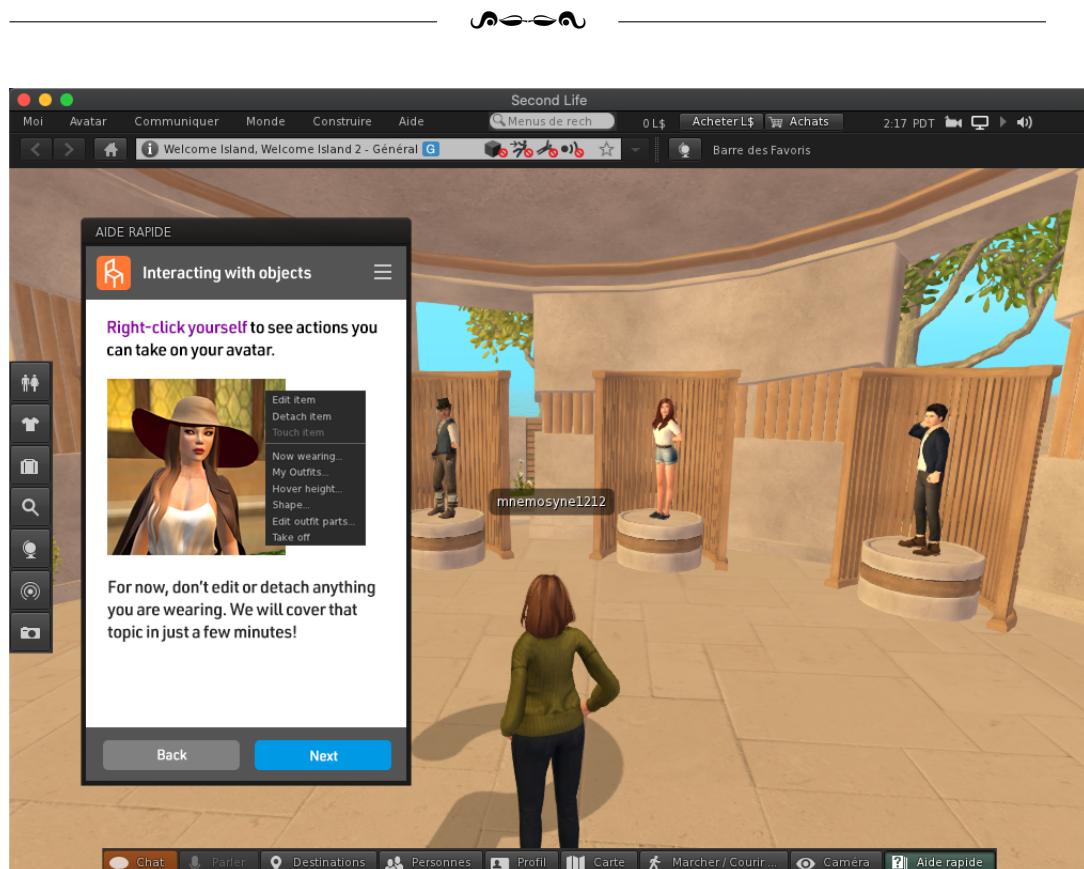
4.1 Quelques Mots sur les Mondes Virtuels : Second Life, Fortnite, Roblox, Blockchain et Métavers(e)

Le 24 avril 2020, en pleine pandémie, le rappeur Travis Scott (Astronomical) donnait un concert virtuel géant devant plus de 12 millions d'internautes dans le jeu Fortnite (Battle Royale) ou encore citons celui d'Ariana Grande (Rift Tour) en août 2021. Outre ces concerts dans celui-ci (Live Events), il y eut notamment l'initiative de Ferrari pour sa 296 GTB, les apparitions du basketteur LeBron James et du footballeur Neymar. Fortnite est aussi un monde de collaboratif où une grande communauté de développeurs intéragissent et proposent des améliorations dans le fonctionnement du jeu. Des groupes musicaux comme par exemple Nightwish se sont produits en mars 2021 sur la plateforme **Burst Live** (Musicverse), Twenty One Pilots en septembre 2021 sur Roblox ou encore les Brit Awards en février 2022 sur cette même plateforme.

Né de la fusion entre les termes "*meta*" (signifiant "au-dessus de", "au-delà de") et "*universe*" (univers en anglais), le *metaverse* (métavers) est défini comme un espace virtuel partagé, immersif, collectif et collaboratif. Ce concept vise à créer une réalité virtuelle physique. Vu comme le successeur d'internet par la majorité des acteurs économiques dans le secteur des technologies, le métavers allie réalité augmentée et espaces virtuels.

Une des premières plateformes que nous pourrions appeler "métavers" fut certainement **Second Life**, créée en 2003 par la société Linden Lab où de grandes sociétés s'implantèrent pour présenter et vendre leurs produits voire des universités qui diffusèrent des conférences et des formations en ligne.

CHAPITRE 4. MONDES VIRTUELS IMMERSIFS ET PLATEFORME COLLABORATIVE



- Un espace de rencontres sur Second Life -

Exercice 4.1 (Pour Réfléchir)

Après avoir téléchargé et installé le viewer Second Life (ou celui de Firestorm) et s'être inscrit.e gratuitement sur la plateforme :

⌚ Déplacez-vous (téléportez-vous (TP)) dans le grid et essayez de rencontrer et de converser en privé avec un.e ami.e connu.e de l'IED (dans le Café Sympa ou ailleurs)

✓ Vous pouvez remplir sommairement votre bibliographie (réelle ou virtuelle) et prendre une photo de votre avatar avec le plus bel effet glow possible ...

✓ Pour vous aider, vous pouvez visionner le tutoriel ci-dessous (QR-Code).



Second Life - le Guide du Débutant sur YT)



Après Second Life, la société de jeu en ligne **Roblox** fut l'une des premières à avoir développé le métavers en construisant des mondes virtuels communautaires afin de découvrir des jeux conçus par d'autres joueurs. Ils ont tous pour caractéristiques que ce sont des mondes immersifs où l'utilisateur possède une identité au sein d'une civilisation virtuelle, un système économique propre (ce sont souvent des crypto-monnaies), une accessibilité illimitée et bien évidemment, une latence très faible de l'interface de jeu car le métavers se veut synchrone, sans limite et diversifié.

Le concept de métavers (ou metaverse) est ainsi devenu familier du grand public depuis que la société Méta de Mark Zuckerberg a commencé à en faire la promotion à l'été 2021. Ce concept de métavers est apparu la première fois dans un ouvrage de science-fiction : *le Samouraï virtuel* de Neal Stephenson (1992), où le héros vit une double existence dans un univers virtuel en 3D. Mark Zuckerberg, le PDG (SEO) de Méta a défini le métavers ainsi lors d'une interview donnée au magazine "The Verge" comme étant "un internet incarné, où, au lieu de voir du contenu, vous faites partie de ce contenu" (cf Revue Elephant N37 de Janvier 2022). Le métavers n'est pas constitué par une seule entreprise, mais par une grande partie de l'industrie du web. Il se prolonge avec les *NFT(s)* (Non Fongible Tokens) ou jetons non fongibles, titres de propriétés numériques fondés sur la *block-chain* (chaîne de blocks) qui permet d'identifier et de tracer la valeur d'un bien (voir par exemple l'achat de parcelles de terrain virtuel sur **Decentraland** et le paiement en crypto-monnaie numérique Mana (Ethereum)). Dans l'avenir proche, l'**Open Metaverse Interoperability Group** (OMI) ambitionnerait de devenir le langage voire l'OS du métavers...



Decentraland sur Bit2Me Academy)



D'autres entreprises issues des GAFAM s'intéressent fortement au métavers; c'est le cas d'Apple, avec le développement d'un casque VR immersif dédié (Apple Glass) ou d'un RealityOS, de Meta avec l'Oculus Quest 2 (Horizon World, Venues ou VRChat), de Microsoft **Mesh** qui s'intègre à Teams, Nike avec Roblox (Nikeland) ou encore **RTFKT**, Adidas avec l'achat de terrains sur SandBox (PixOwl) tout comme Carrefour, pour ne citer qu'elles.

Metavers sur JustGeek et InCyber



Il était tout naturellement prévu de réaliser quelques exercices autour du métavers ou de la blockchain. Compte tenu de l'évolution future de ces technologies et de la difficulté de mise en oeuvre d'exercices simples à réaliser, ce sera pour la prochaine mouture du cours...

4.2 Découverte de la Plateforme Collaborative Streamlit

La Plateforme Streamlit, créée en 2018, est définie par **ses créateurs** (Adrien Treuille, Amanda Kelly et Thiago Teixeira) comme "le moyen le plus rapide de créer et de partager des data apps". En plus de ses qualités pour la datascience, pourquoi s'intéresser ici à cet outil Python? C'est parce que les API de Streamlit sont très riches et nous permettent d'afficher des textes simples, des checkbox, des sliders, des graphiques, des images, des



cartes, du code, ... et des buttons... Et tout cela, avec très peu de lignes de commandes. Cette plateforme open-source , achetée par le Data Cloud Snowflake en mars 2022, permet aux développeurs et aux data-scientists de créer et de partager des applications xeb, et ce, rapidement et de manière itérative, sans avoir besoin d'être un expert en développement frontal. Streamlit compte à la date du cours plus de 8 millions de téléchargements et plus de 1,5 million d'applications ont été créées à l'aide de son framework Python.

Qu'est-ce que Streamlit, une introduction



4.2.1 Installation de Streamlit (Python >= 3.7)

L'installation de Streamlit est assez simple en utilisant à nouveau PIP :

Installation via Pip

```
pip install streamlit
```

Afin de vérifier la bonne installation de la librairie, il suffit, comme l'indique la documentation, de lancer dans le Terminal :

test de l'installation

```
streamlit hello
```



La Documentation de la Librairie Streamlit



Résultat du Lancement de Streamlit

Welcome to Streamlit. Check out our demo in your browser.

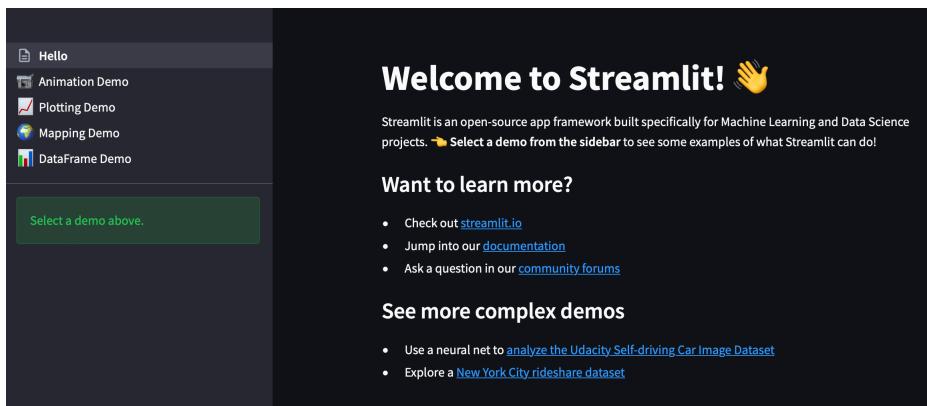
Local URL: <http://localhost:8501>

Network URL: <http://192.168.1.17:8501>

Ready to create your own Python apps super quickly?
Head over to <https://docs.streamlit.io>

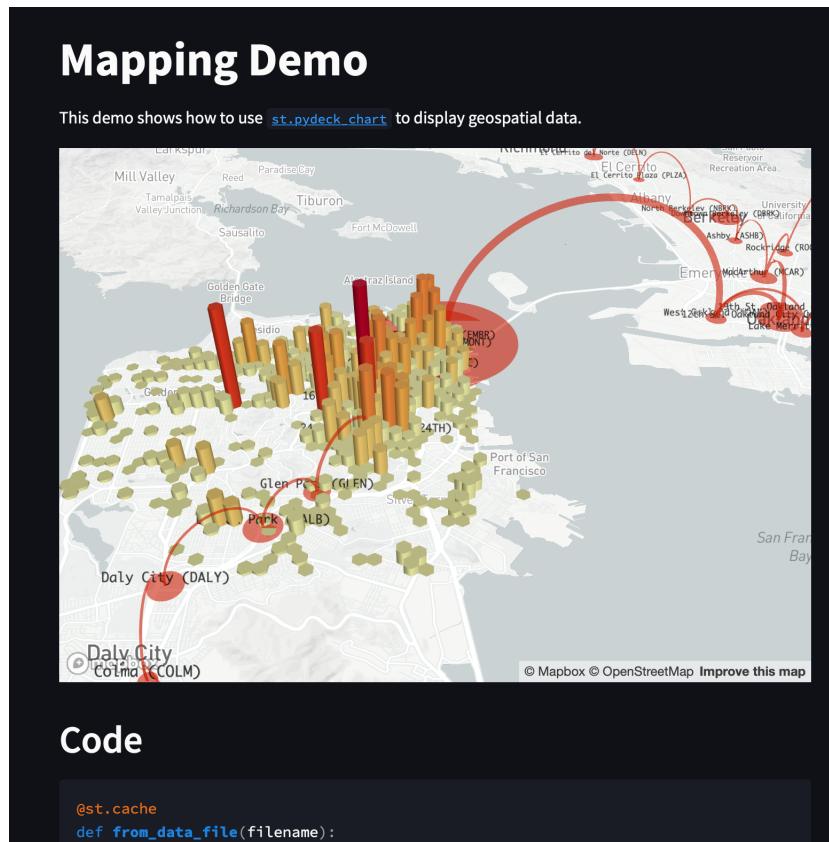
May you create awesome apps!

Si votre navigateur et votre système sont correctement paramétrés, ce premier doit se lancer et afficher la page suivante (Fig.1) :



- Fig.1 - Page Demo de Streamlit -

Dans le cas contraire, il suffit de saisir l'URL indiquée <http://localhost:8501>. Essayons ensuite de tester tous les onglets proposés et afficher les codes correspondants (Fig.2) :



- Fig.2 - Mapping Demo -

4.2.2 Un Tout Premier Programme

Streamlit nous offre la possibilité d'utiliser un grand nombre de "widgets" comme des boutons, des cases à cocher, des zones de textes,... et bien d'autres. Dans votre éditeur de textes préféré saisissez le code Python suivant :

Résultat du Lancement de Streamlit

```

1 import streamlit as st
2 prenom = st.text_input('Quel est votre prénom ?')
3 message = st.text_input('Quel est votre message ?')
4 st.write(message, " ", prenom)

```

Si vous avez lancé "streamlit demo", il suffit d'arrêter le serveur avec la sé-

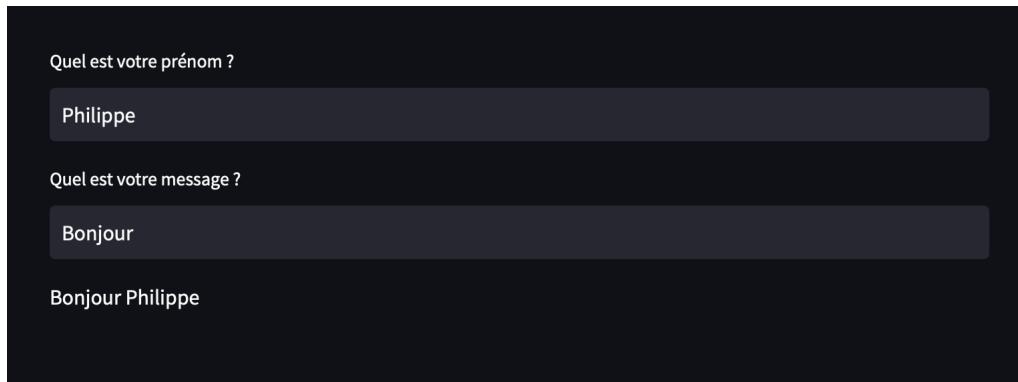


quence de touches "Ctrl-C" habituelle puis après avoir saisi et sauvegardé le code ci-dessous (ici en "bonjour.py") de relancer Streamlit avec notre code :

Lancement de Streamlit

```
streamlit run bonjour.py
```

Comme nous le voyons dans la (Fig.3), nous obtenons deux champs de saisie de texte dont leur contenu respectif sera concaténé l'un à l'autre et sera affiché par la fonction "write" :



- Fig.3 - Notre Premier Programme -

En suivant les tutoriaux de la Documentation et ceux de l'article introduc-tif (cf QR-Code), amusez-vous à tester l'un ou l'autre widget proposé par ce framework...

Une spécificité qui nous intéresse particulièrement dans le cadre de ce cours est la possibilité de lancer notre code directement depuis l'un ou l'autre repo Github, le nôtre ou tout autre repo public :

Lancement de Streamlit depuis un Repo Github

```
streamlit run https://github.com/Mon-Repo/Mon-Titre/blob/
master/bonjour.py
```

Egalement, de compléter notre installation de Streamlit par de nombreux composants complémentaires : **Aggrid**, **Echarts**, **Folium**, **SpaCy**, ... pour ne



citer que les principaux.

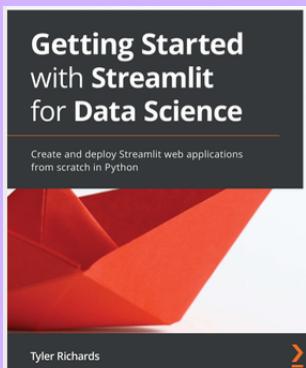
Les Ressources de La Communauté de Streamlit sur Github



4.3 Quelques Liens pour Aller plus Loin...

Quelques Tutoriaux et Livres à Explorer...

- (1) Le livre de Tyler Richards chez Packt Pub, "Getting Started with Streamlit for Data Science", 2021 :



!

:
:

- (2) La Chaîne YT de JCharisTech, qui comporte beaucoup de vidéos de débutant.e à expert.e (à voir) :



- (3) Les Tutoriaux par les Concepteurs de Streamlit toujours sur YT :



- (4) Le GitHub de 30DaysOfStreamlit qui présente un défi de 30 jours pour vous permettre d'apprendre, de créer et de déployer des applications Streamlit

!



- (5) Un Petit Guide du Débutant de Streamlit (faire une recherche Google pour en trouver beaucoup d'autres) :



4.4 Les Exercices du TP

Pour ce quatrième et dernier TP, l'**exercice 4.2** est à Rendre. L'**exercice 4.3** qui vous demandera un peu de connaissances en Cryptographie et dans les "langages exotiques" est pour un Bonus orienté Mathématiques assez facile.

Exercice 4.2 (A Rendre)

En utilisant la plateforme Streamlit et votre espace (Repo) Github :

- » (1) **Insérez une photographie** (de vous, de Mnemosyne (cf GIL), de la personnalité de votre choix...) dans votre application Streamlit et **générez un formulaire complet** de manière à éditer (changer) un maximum de métadonnées EXIF (EXchangeable Image



file Format) de cette image. **Déposez votre code-source dans votre espace Github** et lancez votre application en utilisant l'URL de celui-ci.

⌚ (2) **Modifiez les données GPS** (GPS tags) de votre photographie afin qu'elles correspondent à l'emplacement où vous vous trouvez actuellement.

✓ Pour vous aider, il existe un grand nombre de librairies Python à utiliser ([Exif](#), [ExifRead](#), la méthode `getexif()` de la Librairie PIL (Pillow),...).

⌚ (3) **Affichez** dans votre application Streamlit **les coordonnées GPS** saisies dans l'image sur une carte du pays correspondant à celles-ci.

⌚ (4) Toujours dans une même application, **dans une autre carte (ou plusieurs selon)**, affichez sous la forme de **POI(s)**, tous les endroits où vous vous êtes rendus (voyages, lieux d'habitation, ...) en les joignant entre eux. (Si vous n'avez pas ou peu voyagé, ajoutez les POI(s) des destinations de rêve que vous aimeriez visiter.)

✓ Pour vous aider, vous pouvez utiliser le composant Folium.

✓ N'oubliez pas d'illustrer vos réponses avec des C.E(s) illustratives, d'expliquer votre méthodologie et d'insérer et joindre vos codes python commentés.

Exercice 4.3 (Bonus)

Après avoir décodé ces informations "ésotériques" :

⌚ **Réalisez** cette première consigne :

```
>++++++[<++++++>-]<++.>+++++[<+++++>-
]<.->+++++[<++++++>-]<.->+++++[<-
-->-]<---.>++++[<++++>-]<-->+++++[<-
-->-]<---.>+++++[<++++++>-
]<+++-.---.>+++[<-->-]<.>+++++[<-->-]<--
->+++++[<++++>-]<-->+++++[<++++>-
]<..--.--.>++++[<++++>-]<++.>+++[<-
>-]<-.+++++->+++++[<-->-
]<++.>+++++[<++++>-]<++.>+++++[<++++>-
]<--.->+++[<-->-]<++.--.>+++[<++>-]<++.-.
--.>+++[<++>-]<++.>+++++[<-->-]<--
.+++++++.>+++++[<++++++>-]<+++.>+++[<-->-
]
```



```

]<-.>+++++++-[<---->-]<-.>+++++++-[<+++++++->-
]<--.>+++++++-[<+++++++->-
]<++++.>+++++++-[<---->-]<--._____
.>+++++++-[<---->-]<--.>+++++++-[<+++++++->-
]<.++++.>++++[<-->-]<.>+++++++-[<---->-]<--
.>++++[<++++>-]<++++._____.++++.>++++[<-->-]<-
.>+++++++-[<+++++++->]<-.+++++.>+++++++-[<-->-]<--
]<--.>+++++++-[<+++++++->]<++++._____
--.>+++[<-->-]<.>+++++++-[<---->-]<--
.>+++++++-[<+++++++->]<+.>++++[<++++>-
]<++++.-.>++++[<-->-]<++.>+++++++-[<---->-
]<++++.>+++[<-->-]<.>+++++++-[<+++++++->-
]<.++++.>+++[<-->-]<--.>+++++++-[<---->-
]<--]<+++++++->+++++++-[<+++++++->-
]<+.>++++[<-->-]<+++.-.+++++.--.>++++[<++++>-
]<--.>++++[<-->-]<+.+++++.--.>+++++++-[<-->-]<--
].>+++++[<++++>]<--.+++++.>+++++++-[<---->-]<-
.>+++++++-[<+++++++->]<--._____.>+++++++-[<-->-]<--
]<--.>+++++++-[<+++++++->]<-.++.>++++[<-->-]<-
->-[<+++.+++++++.--.---.>++++[<++++>]<--
].>+++++++-[<---->-]<-.>+++++++-[<+++++++->-
]<--.--.++..--.+++++-.>+++++++-[<---->-
]<+++.>+++++++-[<+++++++->]<+++.+.>+++++++-[<-->-]<--
]<--.>+++++++-[<---->-]<--.>+++++++-[<---->-
]<++++.>+++++++-[<+++++++->]-<+++++++-[<---->-
]<----->+++++++-[<---->-]<--.>+++[<---->-
]<+++++++-[<+++++++->]<--.>+++[<---->-]<-
]<++.--.>++++[<---->-]<--.>+++++++-[<---->-]<-
.>+++++++-[<+++++++->]<+++++.+.>+++++++-[<---->-
]<--.>+++++++-[<+++++++->]<--._____.>+++++++-[<---->-
]<--]<--.>+++++++-[<+++++++->]<--._____.>+++++++-[<---->-
]<--.>+++++++-[<---->-]<--.>+++++++-[<---->-
]<--.>+++[<-->-]<--.++.>+++++++-[<---->-]<--
].>+++[<-->-]<--.++.>+++++++-[<---->-]<+
].>+++[<---->-]<++.>+++++++-[<---->-]<-
--.>+++[<---->-]<++.>+++++++-[<---->-]<-
--.>+++[<-->-]<.>+++++++-[<---->-]<-
--.>+++[<-->-]<.>+++++++-[<---->-]<-

```



```
.>++++++[<++++++>-]<+++.-.>++++[<++++>-
]<++.>+++[<-->-]<+.>++++++[<--->-]<--
.>++++++[<++++++>-]<+++.+.>++++++[<---
->-]<--.>++++++[<++++++>-]<--.>++++[<---
->-]<+++.++++++.>+++[<+++>-]<..>+++[<-->-]<+.
.>++++[<+++>-]<--.>+++[<-->-]<-.++.>++++++[<---
->-]<--.>++++++[<++++++>-]<++.>+++[<-->-]<--.
.>++++++[<--->-]<--.>++++++[<++++++>-
]<-.>+++[<-->-]<+.>+++[<+++>-]<--.>++++++[<---
->-]<-.>++++++[<++++++>-]<--.>++++++[<---
->-]<--.>++++++[<++++++>-]<--..>+++[<-->-]<-
.++.-.>+++[<+++>-]<+.>+++[<-->-]<-.>+++[<+++>-
]<+++.>+++[<-->-]<+.>+++[<+++>-]<--.>++++++[<---
->-]<--.>+++[<-->-]<++.
```

⌚ Puis cette seconde consigne (suite) :

```
>++++++[<++++++>-]<+++.>++++[<++++>-
]<+++.>+++[<+++>-]<-.++++.>++++++[<---
->-]<-.>++++++[<++++++>-]<+++.____-
.>++++++[<--->-]<++.>++++++[<++++++>-
]<+++.++++.____-.>+++[<+++>-]<+++.++.>++++++[<---
->-]<+.>++++++[<++++++>-]<+++.>+++[<---
->-]<+.>+++[<+++>-]<+++.___.>+++[<-->-
]<+.>++++++[<--->-]<++++++.>+++[<-->-
]<--.>++++++[<++++++>-]<+.>+++[<+++>-
]<-.++.>++++++[<++++++>-]<--
.>++++++[<--->-]<+++.>++++++[<---
->-]<-.>++++++[<++++++>-]<--
.>++++++[<--->-]<--.>++++++[<++++++>-
]<--.>+++[<+++>-]<--._____.>+++[<+++>-
]<.>++++++[<--->-]<-.>++++++[<++++++>-
]<+++.-.>+++[<+++>-]<+++.>+++[<-->-]<.>+++[<+++>-
]<++.>+++[<-->-]<.>++++++[<++++++>-
]<+++.>++++++[<--->-]<--.>++++++[<++++++>-
].>++++++[<++++++>-]<+.>++++[<++++>-
]<--.>+++[<-->-]<--.>++++++[<--->-]<--
.>++++++[<++++++>-]<-.>++++++[<--->-]<--.
```



```

]<+++++.—.>+++[<-->-]<—.++++++.-.>++++++[<—
—>-]<++.>++++++[<++++++>-]
]<-.+++++.>++++++[<——>-]<—
.>++++++[<++++++>-]<-.>+++[<—>-
->-]<++.>+++[<++>-]<+.>+++[<—>-
]<+.>+++[<++>-]<.+++++.>++++++[<——>-
]<—.>++++++[<++++++>-]<+++++.>+++[<++>-
]<.>++++++[<——>-]<++.>++++++[<++++++>-
]<—.—.+++++.>+++[<—>-]<.>++++++[<—>-
]<+++++.>+++[<—>-]<—.>++++++[<++++++>-
]<+.++++..+++.—.++++.—.>+++++[<++++>-
]<—.>++++++[<——>-]<—
.>++++++[<++++++>-]<—.—.>++++++[<—
—>-]<—.>++++++[<++++++>-]<—
.-+++++.++++.—.>++++++[<——>-]<-
.>++++++[<++++++>-]<++.+.>++++++[<—
—>-]<—.>++++++[<++++++>-]<—.>+++[<—
->-]<—.>++++++[<++++++>-]<—.>+++[<—
->-]<++.+++++.>+++[<++>-]<..>+++[<—>-
.>+++[<++>-]<—.>+++[<—>-]<—
.>+++[<++>-]<—.>++++++[<—>-]<—.>+++[<—>-
]<+++.+++.>+++[<—>-]<+++.>+++[<++>-
]<—.—.++[<—>-]<—.—.>+++[<++>-]<—
.+++++.>++++++[<——>-]<—.

```

⌚ Et Encore... (suite) :

```

>++++++[<++++++>-]<+++.>+++[<++>-
]<+++.>+++[<++>-]<—.++++.>>++++++[<—
—>-]<—.>++++++[<++++++>-]<+++.—
.>++++++[<——>-]<+++.>++++++[<++++++>-
]<+.>+++[<++>-]<+++.—.>>+++[<—>-
]<+++.>++++++[<——>-]<—.>++++++[<++++++>-
]<+++.++++.—.>>+++[<++>-]<+++.++.>++++++[<—
—>-]<+.>++++++[<++++++>-]<+++.>+++[<—
->-]<+.>+++[<++>-]<+++.—.>>+++[<—>-
]<+.>++++++[<——>-]<++++++.>+++[<—>-
.>++++++[<++++++>-]<+++.—.>>+++[<++>-
]<+.>+++[<—>-]<++.>+++[<++>-]<+—
.—.>++++[<++++>-]<—.>++++++[<——>-]<—
.—.>++++++[<++++++>-]<—.>+++[<—>-]<—

```



```

.>+++++++-[<---->-]<-.>+++++++-[<+++++++->-
]<++.---.-..---.>+++++++-[<---->-]<--
.>+++++++-[<+++++++->-]<++++.+>++++[<++++>-]<-
.>+++++++-[<---->-]<-.>++++[<++++>-]<++.-.>++++[<-->-
->-]<.>+++++++-[<+++++++->-]<-.>++++[<-->-
]<++.+++++++-.-.>+++++++-[<+++++++->-
]<++++.+++++++->+++++++-[<+++++++->-
]<++++.>>++++[<-->-]<++.>>++++[<++++>-]<-
.>+++++++-[<---->-]<-.>+++++++-[<+++++++->-
]<++++.>>+++++++-[<+++++++->-
]<----.>>+++++++-[<---->-]
]<+++++++-+.+++++.++++.>+++[<-->-]<-.>+++[<+++>-
]<++.---.>++++[<++++>-]<-.>+++++++-[<---->-]<-
.>+++++++-[<+++++++->-]<++++.+>+++++++-[<-->-
->-]<--.>+++++++-[<+++++++->-]<--.---.>+++++ [<-->-
>-]<--.>+++[<-->-]<--.>+++++++-[<+++++++->-]<-
.+++++.>+++[<-->-]<--.>+++[<+++>-]<+.>+++++++-[<-->-
]<-->-]<--.>+++++++-[<+++++++->-]<++++.--
.>++++[<++++>-]<--.+++++.>+++++++-[<-->-
->-]<--.>+++++++-[<+++++++->-]<++++.--
.>+++++++-[<---->-]<++.>+++++++-[<+++++++->-
]<++.>>+++[<-->-]<++.-.>+++[<+++>-]<++.-.>+++[<-->-]<-
.>+++++++-[<---->-]<--.>+++++++-[<+++++++->-
]<--.>+++[<-->-]<--.>+++++++-[<---->-]<-
].>+++++++-[<+++++++->-]<++.--.-..-->-
.>+++++++-[<---->-]<--.>+++++++-[<+++++++->-
]<+++++.>+++[<---->-]<--.>+++++++-[<+++++++->-
]<--.>+++++++-[<---->-]<--.>+++++++-[<---->-]<-
.>+++++++-[<---->-]<--.>+++++++-[<---->-]<-
]<----.>>+++++++-[<---->-]
]<----.>>+++++++-[<---->-]<--.>+++++++-[<---->-]<-
]<++.---.>++++[<++++>-]<-.>+++++++-[<---->-
>-]<--.>++++[<++++>-]<--.>+++++++-[<---->-]
]
```



```
]<++++++.>++++++[<++++++>-]<.>+++++[<+++++>-
]<.>>++++[<-->-]<.>++++++[<--->-]<---
.>++++++[<++++++>-]<+.>>++++[<-->-
]<+++.+++++++.>+++[<-->-]<--.>++++[<+++>-
]<+-.-.++++.>++++[<-->-]<.>+++++[<++++>-]<---
.>++++++[<--->-]<++++.>++++++[<++++++>-
]<---.---.+++++.-.>++++++[<--->-]<-
.>+++++[<+++++>-]<---.
```

⌚ Et Encore... (fin) :

```
>++++++[<++++++>-]<---.>+++++[<+++++>-
]<+.+++++.>++++++[<++++++>-]<---
.>++++++[<--->-]<+.>>++++[<-->-
]<+++.>+++++[<+++++>-]<--.>++++++[<---
-->-]<---.>++++++[<++++++>-]<++++.--
.>++++[<++++>-]<--.++++.>++++++[<---
>-]<--.>++++++[<++++++>-]<++++.---
.>++++.->++++[<-->-]<++.>>++++++[<--->-]<---
->++++++[<++++++>-]<+.>>++++[<++++>-]<---.-
---.---.->++++[<++++>-]<+++.>>++++++[<---
>-]<--.>++++++[<++++++>-]<++++.>>+++[<-->-
]<+++.---.>+++[<-->-]<.>++++++[<--->-]<---
->++++++[<--->-]<--.>++++++[<--->-
]<+++.---.>++++++[<++++++>-]<---.
->++++++[<--->-]<--.>++++++[<--->-
]<--]<++.>>++++++[<++++++>-]<-.++.>+++[<-->-
]<.>+++[<+++>-]<+.>>+++[<-->-]<++.>>++++++[<---
>-]<--.>++++++[<++++++>-]<++++.>>+++[<+++>-
]<.>++++++[<--->-]<++.>>++++++[<++++++>-
]<--.---.+++++.>+++[<-->-]<.>++++++[<---
-->-]<--.>++++++[<++++++>-
]<.-+++++.>+++[<-->-]<--.>++++++[<--->-
]<+++++++.>++++++[<++++++>-]<---
.+-.+++.--.>++++[<-->-]<++.>>++++++[<---
-->-]<--.>++++++[<++++++>-
]<.-+++++.>++++[<-->-]<.>++++++[<--->-]<---
.>++++++[<--->-]<--.>++++++[<--->-
]<++++++[<--->-]<--.>++++++[<--->-
```



```
[<++.---...---.>++++++[<--->-]<---  
.>>++++++[<++++++>-]<++++.+.>++++++[<---  
->-]<---.>++++++[<++++++>-]<+++.--.+++++-  
.>>++++++[<--->-]<-.>++++++[<++++++>-]  
]<---.---.>++++[<++++>-]<.->++++++[<++++++>-]  
->---->-[<++++++>-]<--->++++++[<---->-]  
.>>----[<-->-]<.->++++[<++++>-]<.>++++[<-->-]  
]<+++.>++++[<++++>-]<.->++++++[<---->-]  
]<-.>++++++[<++++++>-]<+++++.>+++[<+++>-]  
]<.+++++.>+++[<-->-]<.-.-->++++[<++++>-]<--  
.+.>++++++[<---->-]<.->++++++[<++++++>-]  
]<--.>++++[<-->-]<+++.>++++[<++++>-]<+++.+--  
.>>++++[<-->-]<+++.+++++++.++++++.>++++++[<--  
--->-]<-.>++++++[<++++++>-]  
]<+++++.>++++[<++++>-]<.->++++++[<---  
->-]<--.>++++++[<++++++>-]<++++++[<++++++>-]  
]<+++++.>++++[<---->-]<--->----.--  
->+++[<+++>-]<++.>++++++[<---->-]  
]<+++++.>++++++[<++++++>-]<---  
.>>++++++[<---->-]<+++.>+++[<+++>-]<--  
.++++.-.++.-.--.
```

- ✓ Les plus audacieux.cieuses pourront déposer leur application sur [share.streamlit.io](#).
- ✓ Essayez d'utiliser un maximum de fichiers ou de documents en ligne en reprenant les URLs dans votre application.
- ✓ Comme à l'habitude, insérez et joignez les codes-sources et illustrez votre méthodologie par de nombreuses C.E(s) illustratives.

4.5 Et pour Clore ces Notes de Cours...

En guise de conclusion provisoire, je vous joins, si vous ne l'avez pas déjà, l'adresse du [GitHub Student Developer Pack](#) qui vous permettra d'utiliser un grand nombre d'applications, d'IDE, et de logiciels en version éducation et gratuite (plus d'une trentaine). Les enseignants pourront également profiter de Github Team, de la Github Classroom, etc.