



Récolte et nettoyage de données automatisées (Linux)

Tony DE CARVALHO | Damjan GRUJICIC
Benoît LE GOFF | Christophe LOUREIRO

Filière Informatique de Gestion
Classe 601_PT_F

Pour : Jérémie VIANIN & Xavier BARMAZ |
Introduction à l'informatique | 15.12.202

I.	Introduction	1
A.	Résumé exécutif	1
B.	Introduction au projet.....	1
C.	Installation de la Virtual Machine.....	2
D.	Installation de Visual Studio Code	6
II.	Cheminement des fichiers	7
III.	Fonctions principales du code.....	8
A.	Structure du code	8
B.	Constants.py	8
C.	Contact.py	9
D.	File.py	9
E.	Mail.py	11
F.	Main.py	12
G.	Server.py	12
H.	Utils.py	15
IV.	Pistes d'améliorations	16
V.	Conclusion.....	16
VI.	Conclusions personnelles	17
A.	Benoît Le Goff	17
B.	Tony de Carvalho.....	17
C.	Damjan Grujicic	18
D.	Christophe Loureiro	18
	Annexe.....	19
	Table des abréviations	II
	Table des illustrations.....	II

I. Introduction

Ce document a pour but de présenter notre projet Python : *Récolte et nettoyage de données automatisées (Linux)*.

A. Résumé exécutif

Dans le cadre de nos études d'Informatique de Gestion à la HES-SO, et plus spécifiquement de notre cours Introduction à l'informatique, nous avons eu comme mission de réaliser un programme python permettant de récolter et de nettoyer des données via un FTP (protocole de transfert de données). Notre programme doit s'activer automatiquement lorsqu'un fichier se trouve sur le FTP.

Pour le système d'exploitation, nous avons le choix de travailler sous Windows, l'OS proposé par Microsoft ou de nous baser sur Linux, un système d'exploitation open-source, c'est-à-dire, dont le code source est disponible pour tous.

Nous avons opté pour Linux, ce choix nous a paru plus naturel python étant à langage de programmation qui évolue parfaitement dans cet environnement. C'est aussi l'occasion pour les membres du groupe n'ayant jamais travaillé dessus de le découvrir.

Comme tous les membres de notre groupe n'ont pas l'OS Linux installée sur nos machines, il nous a fallu procéder à l'installation du système d'exploitation au travers du machine virtuelle. Les VM nous permettent de simuler un OS sur notre machine dans un environnement clos.

Nous avons décidé d'éditer notre code sur Visual Studio Code qui possède une intégration poussée de Python et une assistance à la complétion. Ayant plusieurs membres du groupe évoluant déjà dans cet éditeur de code, ce choix nous a donc paru la solution la plus logique.

Afin de garantir un suivi de développement optimal nous avons décidé d'évoluer sur la plateforme GitHub. Cette manière de procéder nous a permis de suivre l'avancée du projet tout en regroupant les modifications de codes qui avait été effectuées par chacun.

B. Introduction au projet

Initialement, nous nous connectons aux deux serveurs FTP et vérifions la réussite de la connections. Nous contrôlons ensuite la présence de dossier de travail obligatoire (Erreur et Traité).

Notre script vérifie dans un intervalle de 15 secondes la présence de fichiers à traiter. Un contrôle du type de fichier est effectué.

Ensuite, notre programme devra vérifier que le fichier reçu contient toutes les colonnes d'une structure d'en-tête valide. Dans le cas d'une erreur, le programme doit arrêter le processus et place le fichier dans un dossier ('Erreur') spécifique du FTP. Il doit aussi avertir l'utilisateur par mail.

Si le fichier est valide, le programme doit scanner les lignes contenues dans le fichier une par une et supprimer celles qui contiennent des erreurs. Par exemple, si lors du scan d'une ligne, une colonne contient un Int au lieu d'un String, la ligne entière sera supprimée. Nous effectuons aussi un contrôle de type pour les valeurs de type Date ou Time.

Une fois le nettoyage fini, le programme enverra le fichier nettoyé sur un autre FTP et il enverra un mail à l'utilisateur le notifiant du succès de l'opération ainsi que le nombre de ligne du fichier qui ont été supprimé. Le fichier original est sauvegardé dans un dossier « traité » sur le serveur FTP original.

C. Installation de la Virtual Machine

Afin d'installer un environnement Linux openSUSE sur VirtualBox, il nous a tout d'abord fallu télécharger le DVD Suse image. Nous avons pu télécharger ce fichier de type ISO via le lien suivant : <https://get.opensuse.org/leap/#download>

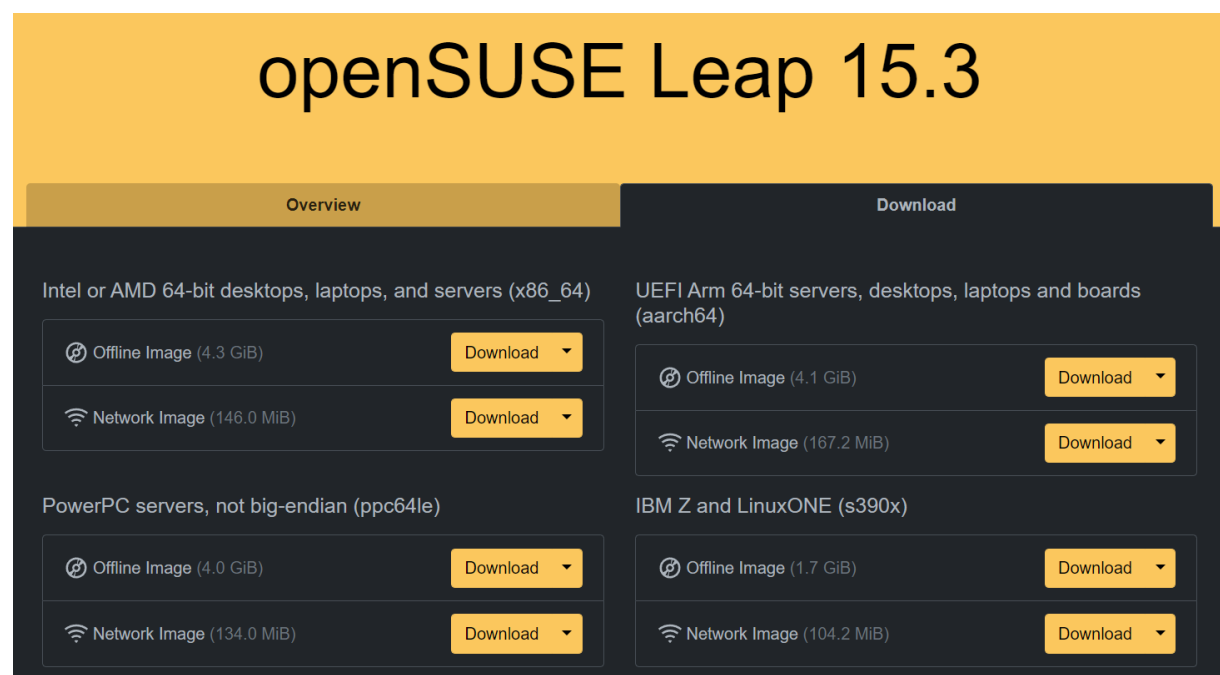


Figure 1 : Site officiel de OpenSuse

Une fois l'ISO téléchargé, nous pouvons enfin installer notre machine virtuelle. Pour cela il nous suffit de lancer le programme VirtualBox et de sélectionner Machine/Nouvelle. Dès lors, une fenêtre s'ouvre et il nous suffit de nommer notre machine, de choisir dans quel dossier elle sera installée ainsi que le type et la version de notre OS, dans notre cas : Linux version : openSuse (64-bit).

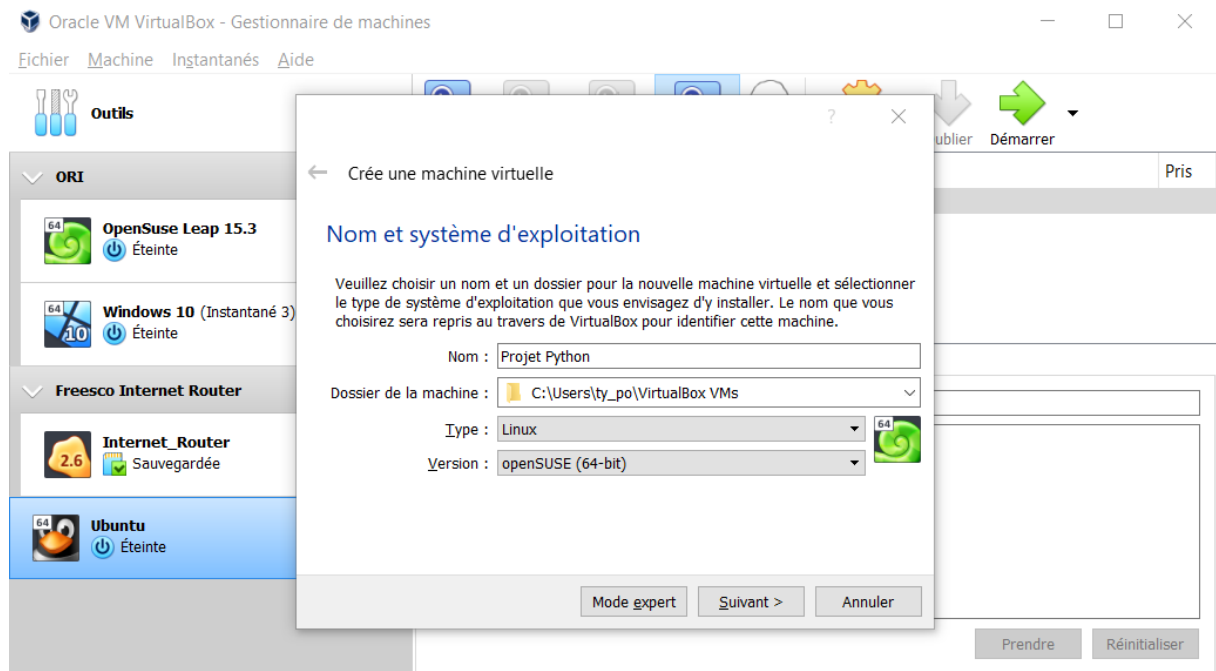


Figure 2 : Création d'une VM dans le programme Oracle VM VirtualBox

Il faut ensuite choisir la mémoire vive allouée à notre machine virtuelle. Pour cela nous nous sommes référés à la documentation fournie durant le cours et avons opté pour une mémoire vive de 2048 Mo de RAM.

Taille de la mémoire

Choisissez la quantité de mémoire vive en méga-octets alloués à la machine virtuelle.

La quantité recommandée est de **1024 Mo**.



Figure3 : Configuration de la VM dans le programme Oracle VM VirtualBox

Il nous a fallu ensuite créer un disque dur virtuel afin de procurer un espace de mémoire allouée à notre machine virtuelle. Pour le type de fichier nous avons choisi le VDI (VirtualBox Disk Image) et nous avons décidé d'allouer l'espace de façon dynamique afin de préserver de l'espace sur nos machines. Pour la taille maximale, nous nous sommes à nouveau référés à la documentation du cours et avons opté pour une taille de 25GO

Emplacement du fichier et taille

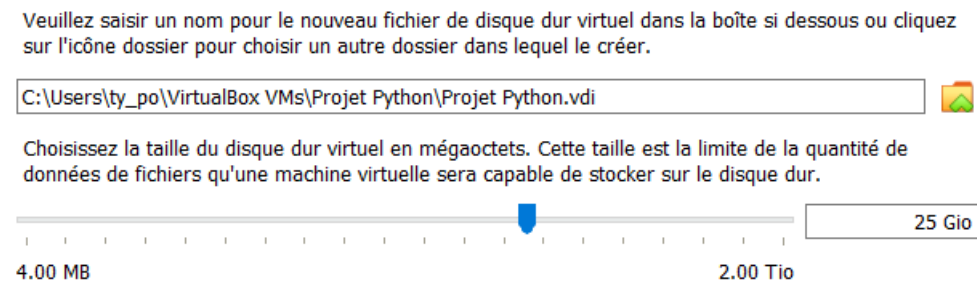


Figure 4 : Emplacements des fichiers de la VM via le programme Oracle VM VirtualBox

Une fois cette étape terminée notre machine virtuelle « Projet Python » apparaît dans notre liste des machines.

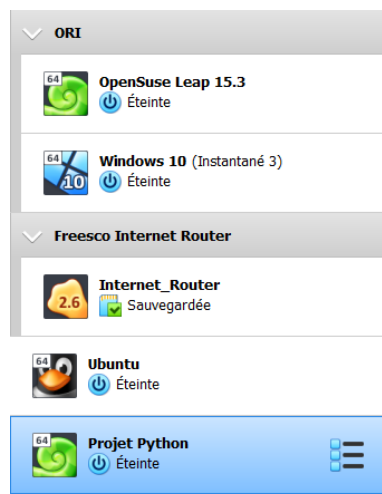


Figure 5 : Machine Virtuelle disponible dans notre programme Oracle VM VirtualBox

Nous devons encore éditer les propriétés de la machine et pour cela il suffit de cliquer sur notre machine, aller sur configuration, stockage, et ajouter un lecteur optique sur le contrôleur : IDE

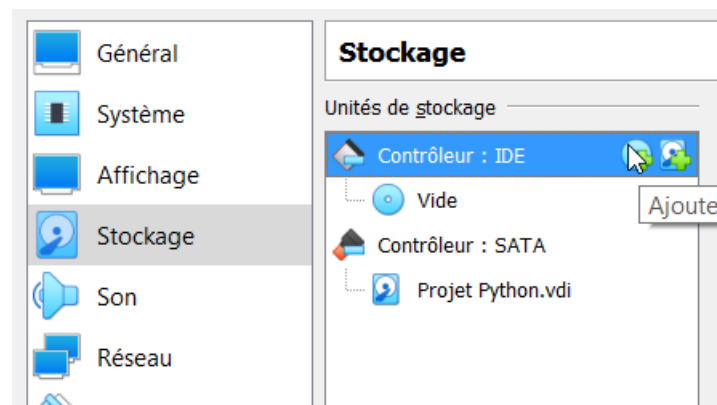


Figure 6 : Editer les propriétés de notre VM dans le programme Oracle VM VirtualBox

Ensuite, nous devons ajouter le fichier ISO préalablement téléchargé et le sélectionner. Après cette opération, il faut se rendre sur Système et choisir l'ordre d'amorçage, dans notre cas l'optique en premier. Sous conseil du professeur après cette action effectuée, pour éviter des mal fonctionnement de la machine nous avons éjecter l'optique et changer l'ordre d'amorçage pour repasser le disque dur en premier. Pour finir nous n'avons plus qu'à configurer le réseau en choisissant le réseau interne.

Nous n'avons plus qu'à lancer la machine virtuelle et configurer notre OS (partitionnement de la mémoire, configuration repositories, compte utilisateur, etc...)

D. Installation de Visual Studio Code

Pour installer Visual Studio Code sur notre environnement Linux, il nous suffit de rentrer cette ligne de code dans le terminal :

```
sudo snap install code --classic
```

L'OS de Linux étant fourni avec une version de python, l'installation de cette dernière n'est pas nécessaire. Nous devons cependant installer l'extension python pour l'éditeur de texte Visual Studio Code.



Figure 7 : Installation de Visual Studio Code dans notre environnement Linux

II. Cheminement des fichiers

Nous avons tout d'abord pensé à un cheminement logique que nous avons imaginé (cf. Annexe, figure 1) pour le traitement de nos fichiers.

Lors du lancement de notre programme, notre programme se connecte au serveur FTP à l'aide d'un *url* et d'un *log in*.

Ensuite, notre code lance une première recherche dans les fichiers existants. Si des fichiers ne sont pas existants, le programme va attendre 15 secondes. Passé ce délai, il va relancer une recherche des fichiers présents jusqu'à ce qu'il puisse en trouver. Dans le cas où des fichiers sont présents sur le serveur, notre programme va lancer une première analyse des fichiers.

Cette première analyse va permettre à notre programme de trouver le type de fichiers. Notre programme recherche les fichiers de type .csv. Dans le cas où ces fichiers ne sont pas dans le format que nous désirons, notre script va lancer une nouvelle attente de 15 secondes et réexécution de recherche jusqu'à ce que les fichiers que nous souhaitons soient trouvés.

Lorsque des fichiers en format .csv sont trouvés, un téléchargement des fichiers va être lancé. Le script va d'abord contrôler les en-têtes (*headers*) du fichier. Supposons que les headers de ces fichiers ne soient pas conformes à une structure correcte attendue, le fichier va alors être déplacé dans un fichier d'erreur et un mail sera envoyé à l'utilisateur pour notifier de l'état d'erreur.

Dès lors que les en-têtes sont conformes à la structure que nous souhaitons, un contrôle au niveau des lignes du fichier va être effectué. En cas de ligne non-conforme aux types attendus, le programme va supprimer la ligne et procéder au contrôle de la ligne suivante, jusqu'à ce que toutes les lignes soient contrôlées et supprimées si besoin.

Au moment où les lignes sont conformes aux attentes que nous nous sommes fixées, le script va créer une colonne *DaySumTot*, qui va faire la somme de tous les en-têtes *DaySum*. Le fichier va être déplacé dans un dossier ./Traité sur l'input FTP, pour finalement être téléchargé sur l'output FTP.

Pour terminer, le programme va envoyer un courriel de succès à l'utilisateur et chercher s'il existe d'autres fichiers à traiter.

Une annexe à ce document décrit le cheminement et le traitement que subit un fichier transmis sur le FTP.

III. Fonctions principales du code

A. Structure du code

Notre code se départage en sept parties distinctes (cf. Annexe, figure 2) :

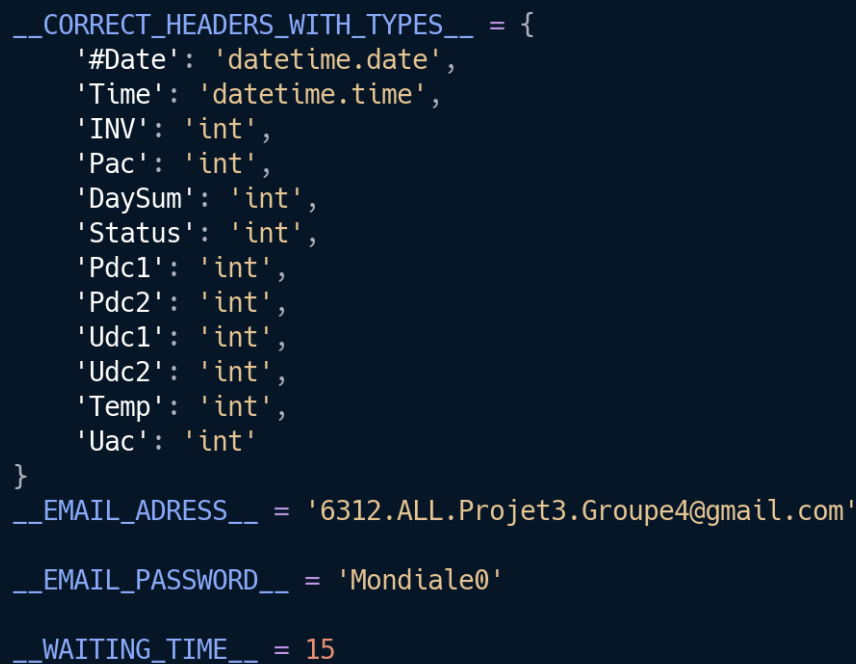
- constants.py, qui contient les variables de script
- contact.py, qui détient les informations de notre interlocuteur
- file.py, qui regroupe les méthodes de traitement des fichiers
- mail.py, qui contient les méthodes liées opération mails
- main.py, où a été rédigé le script principal de notre programme
- server.py, détient les informations de connexion et de traitement des fichiers
- utils.py, a été créé pour mettre en place des méthodes auxiliaires non associées à une classe

Nous allons détailler dans les points qui suivent les différentes étapes parties du code qui ont été mises en place dans le but de pouvoir rendre notre script exécutable et sans erreurs.

Une annexe à ce document décrit la structure du code via un diagramme de classe.

B. Constants.py

Ce fichier contient un dictionnaire avec le nom des headers et leur type respectif attendu. Il contient également le nom de l'adresse électronique et le mot de passe avec lequel les mails de confirmation et d'erreurs seront envoyé.

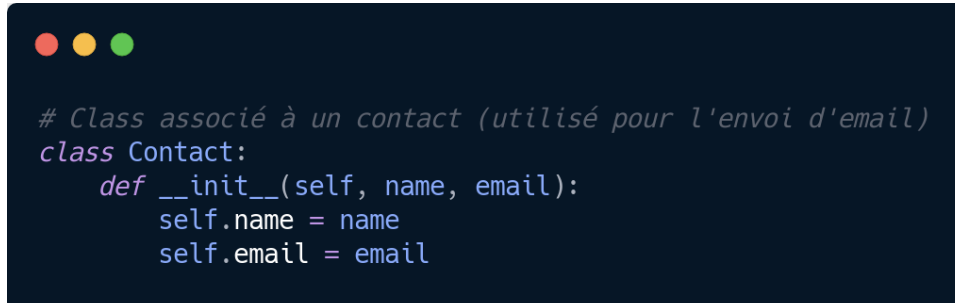


```
__CORRECT_HEADERS_WITH_TYPES__ = {
    '#Date': 'datetime.date',
    'Time': 'datetime.time',
    'INV': 'int',
    'Pac': 'int',
    'DaySum': 'int',
    'Status': 'int',
    'Pdc1': 'int',
    'Pdc2': 'int',
    'Udc1': 'int',
    'Udc2': 'int',
    'Temp': 'int',
    'Uac': 'int'
}
__EMAIL_ADRESS__ = '6312.ALL.Projet3.Groupe4@gmail.com'
__EMAIL_PASSWORD__ = 'Mondiale0'
__WAITING_TIME__ = 15
```

Figure 8 : Capture d'écran de constants.py

C. Contact.py

Cette page contient la classe *Contact*. Elle récupère le prénom et l'adresse électronique du destinataire et crée un profil de l'interlocuteur, qui sera utilisé lors du traitement du fichier pour avertir avec un message en cas de succès ou d'erreur.

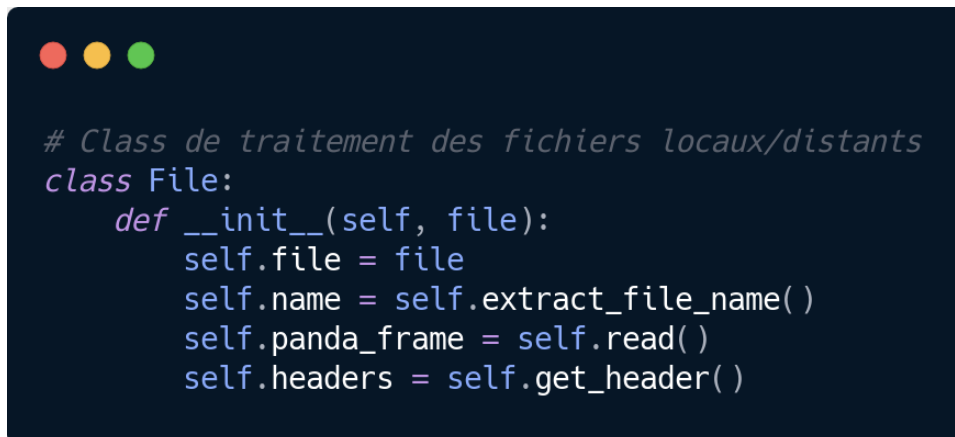


```
# Class associé à un contact (utilisé pour l'envoi d'email)
class Contact:
    def __init__(self, name, email):
        self.name = name
        self.email = email
```

Figure 9 : Capture d'écran de class *Contact* de *contact.py*

D. File.py

La page *File.py* contient la classe *File*. Elle traite les fichiers reçus par plusieurs méthodes.



```
# Class de traitement des fichiers locaux/distants
class File:
    def __init__(self, file):
        self.file = file
        self.name = self.extract_file_name()
        self.panda_frame = self.read()
        self.headers = self.get_header()
```

Figure 10 : capture d'écran class *File* de *file.py*

Les méthodes qui nous intéressent principalement dans notre travail sont les méthodes « *get_headers* » et « *compare_headers*. »

La méthode « *get_headers* » permet de récupérer les headers du fichier et de les stocker dans un tableau.

```
# Récupération des en-tête du fichier csv avec option "brut" pour éviter la
# déduplication de pandas
def get_header(self, type=''):
    panda_columns = self.panda_frame.columns
    if type == 'brut':
        brut_columns = []
        for element in panda_columns:
            brut_columns.append(element.split('.')[0])
        return brut_columns
    else:
        return panda_columns

# Nettoyage des éléments du headers pour éviter les xxx.1, xxx.2, xxx.3 de pandas,
# utilisé dans le cadre de traitement des daySum
def purge_header_item(self, header):
    return header.split('.')[0]

# Récupération des n headers daySum.n
def get_all_daysum_header(self):
    panda_columns = self.panda_frame.columns
    daysum_columns = []
    for element in panda_columns:
        if element.split('.')[0] == 'DaySum':
            daysum_columns.append(element)
    return daysum_columns
```

Figure11 : Capture d'écran de la méthode *get_headers* dans *file.py*

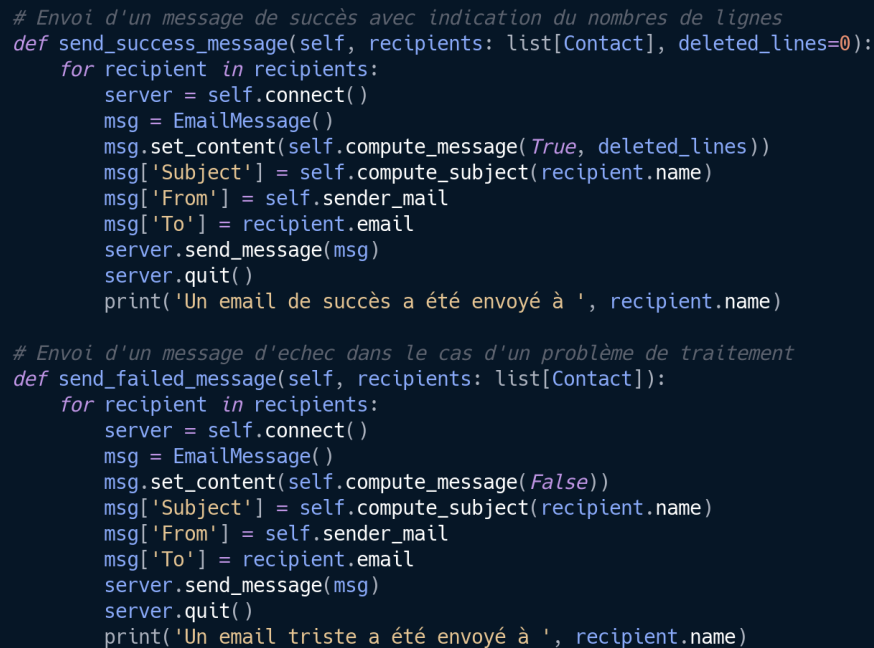
La méthode « *compare_headers* » a pour but de contrôler les en-têtes des fichiers reçus et renvoie donc le résultat de la comparaison entre cette dernière et l'objet `__CORRECT_HEADERS_WITH_TYPES__` contenant le nom de ces en-têtes.

```
# Comparaison des en-têtes avec les valeurs par défaut provenant de constants.py
def compare_headers(self):
    header_check = True
    headers = self.get_header('brut')
    for item in headers:
        if item in __CORRECT_HEADERS_WITH_TYPES__.keys():
            pass
        else:
            header_check = False
            break
    return header_check
```

Figure 12 : Capture d'écran de la méthode *compare_headers* dans *file.py*

E. Mail.py

Cette page contient la classe *Mail* contenant plusieurs méthodes elle va recevoir les informations de la boîte électronique et se connecter à cette dernière. Selon le traitement des données elle peut : soit envoyer un mail indiquant le succès de l'opération et le nombres de lignes supprimées, soit envoyer un message stipulant que les données n'ont pas pu être traitées à la suite d'une erreur.



```
# Envoi d'un message de succès avec indication du nombres de lignes
def send_success_message(self, recipients: list[Contact], deleted_lines=0):
    for recipient in recipients:
        server = self.connect()
        msg = EmailMessage()
        msg.set_content(self.compute_message(True, deleted_lines))
        msg['Subject'] = self.compute_subject(recipient.name)
        msg['From'] = self.sender_mail
        msg['To'] = recipient.email
        server.send_message(msg)
        server.quit()
        print('Un email de succès a été envoyé à ', recipient.name)

# Envoi d'un message d'echec dans le cas d'un problème de traitement
def send_failed_message(self, recipients: list[Contact]):
    for recipient in recipients:
        server = self.connect()
        msg = EmailMessage()
        msg.set_content(self.compute_message(False))
        msg['Subject'] = self.compute_subject(recipient.name)
        msg['From'] = self.sender_mail
        msg['To'] = recipient.email
        server.send_message(msg)
        server.quit()
        print('Un email triste a été envoyé à ', recipient.name)
```

Figure 13 : Capture d'écran des méthodes dans mail.py

F. Main.py

Ce fichier contient la configuration/déclaration initiale des classes principales (Host, Mail, Contact) ainsi que la boucle d'exécution du script principal.

C'est le fichier qui doit être exécuté pour démarrer le script.

```
# Configuration des deux hosts
input_ftp_host = Host('input', 'd73kw.ftp.infomaniak.com',
                      'Cours/Projet3/Groupe1/Input', 'd73kw_projet3_groupe1_i',
                      '6zNr8c9TrHV4')
output_ftp_host = Host('output', 'd73kw.ftp.infomaniak.com',
                      'Cours/Projet3/Groupe1/Output',
                      'd73kw_projet3_groupe1_o', '84JN59cHQbvg')

# Initialisation du serveur
server = Server(input_ftp_host, output_ftp_host)

# Création des contact
benoit = Contact('Benoît', 'benoit.legoff@students.hevs.ch')
christophe = Contact('Christophe', 'christophe.loureiro@students.hevs.ch')
damjan = Contact('Damjan', 'damjan.grujicic@students.hevs.ch')
tony = Contact('Tony', 'tony.decarvalho@students.hevs.ch')
recipients = [benoit, christophe, damjan, tony]

# Connection au serveur et check (connection + présence des dossier de traitement de
# fichiers)
server.connect()
server.check_connection()
server.check_directories()
```

Figure 14 : Capture d'écran de méthodes dans main.py

G. Server.py

La page *Server.py* contient la classe *Host* et la classe *Server*.

La classe *Host* récupère simplement les informations de connexions au FTP qui sont l'url de connexion, l'*username* et le *password*.

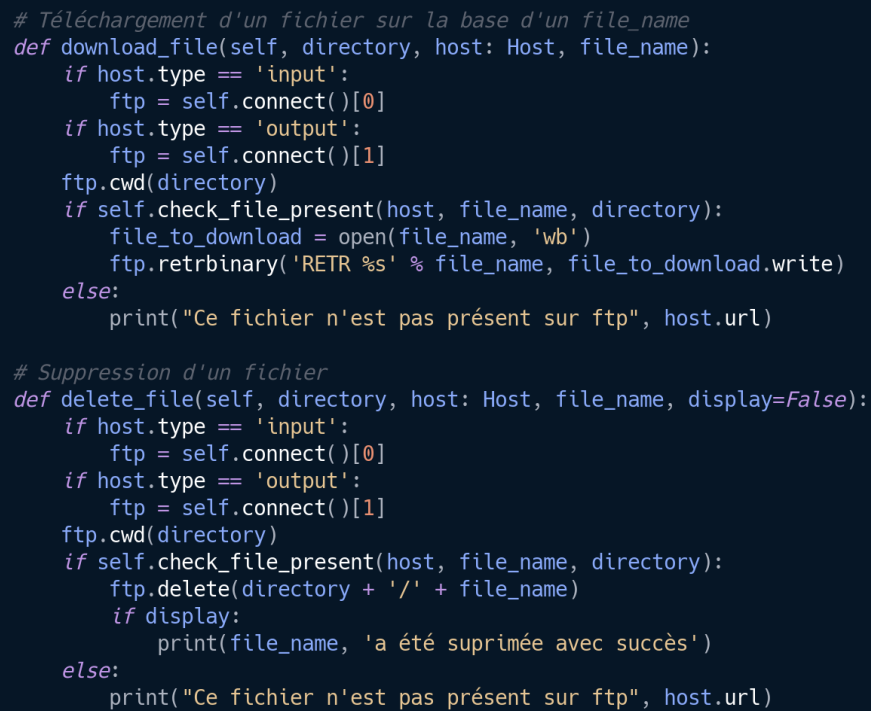
```
# Classe s'occupant de l'enregistrement des host ftp
class Host:
    def __init__(self, type, host, repo, username, password):
        self.type = type
        self.url = host
        self.repo = repo
        self.username = username
        self.password = password
```

Figure 15 : Capture d'écran de server.py

La classe *Server* se charge de récupérer les informations de connexions via la classe *Host*, qui va ensuite se connecter au FTP pour télécharger, charger et supprimer les fichiers ainsi que créer un répertoire de stockage.

Notre classe permet des traitements suivants sur les fichiers :

- Upload d'un fichier vers le FTP dans un dossier spécifique
- Suppression d'un fichier sur le FTP
- Contrôle de présence d'un fichier



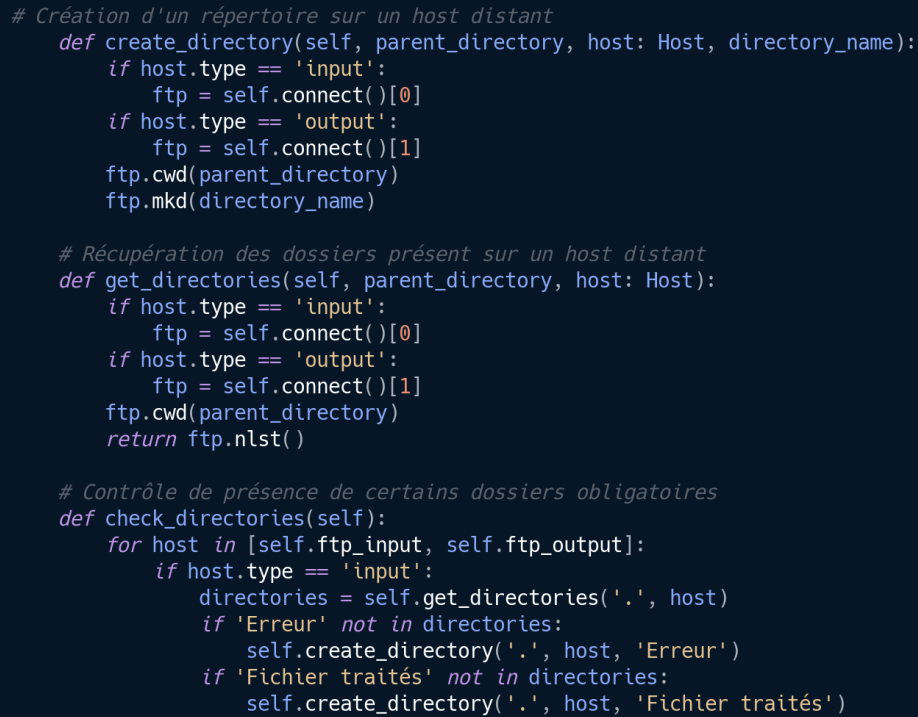
```
# Téléchargement d'un fichier sur la base d'un file_name
def download_file(self, directory, host: Host, file_name):
    if host.type == 'input':
        ftp = self.connect()[0]
    if host.type == 'output':
        ftp = self.connect()[1]
    ftp.cwd(directory)
    if self.check_file_present(host, file_name, directory):
        file_to_download = open(file_name, 'wb')
        ftp.retrbinary('RETR %s' % file_name, file_to_download.write)
    else:
        print("Ce fichier n'est pas présent sur ftp", host.url)

# Suppression d'un fichier
def delete_file(self, directory, host: Host, file_name, display=False):
    if host.type == 'input':
        ftp = self.connect()[0]
    if host.type == 'output':
        ftp = self.connect()[1]
    ftp.cwd(directory)
    if self.check_file_present(host, file_name, directory):
        ftp.delete(directory + '/' + file_name)
        if display:
            print(file_name, 'a été supprimée avec succès')
    else:
        print("Ce fichier n'est pas présent sur ftp", host.url)
```

Figure 16 : Capture d'écran des méthodes de téléchargement et de suppression dans server.py

Nous classer permet aussi d'effectuer les traitements suivants sur les dossiers :

- Création d'un dossier
- Contrôle de présence d'un dossier
- Récupération des dossiers présent sur le FTP

A screenshot of a code editor with a dark blue background and light-colored text. The code is in Python and defines three methods for an FTP client: `create_directory`, `get_directories`, and `check_directories`. The `create_directory` method connects to an FTP server and creates a new directory. The `get_directories` method connects to an FTP server and returns a list of directories. The `check_directories` method checks for the presence of 'Erreur' and 'Fichier traités' directories and creates them if they do not exist. The code is commented in French.

```
# Création d'un répertoire sur un host distant
def create_directory(self, parent_directory, host: Host, directory_name):
    if host.type == 'input':
        ftp = self.connect()[0]
    if host.type == 'output':
        ftp = self.connect()[1]
    ftp.cwd(parent_directory)
    ftp.mkd(directory_name)

# Récupération des dossiers présent sur un host distant
def get_directories(self, parent_directory, host: Host):
    if host.type == 'input':
        ftp = self.connect()[0]
    if host.type == 'output':
        ftp = self.connect()[1]
    ftp.cwd(parent_directory)
    return ftp.nlst()

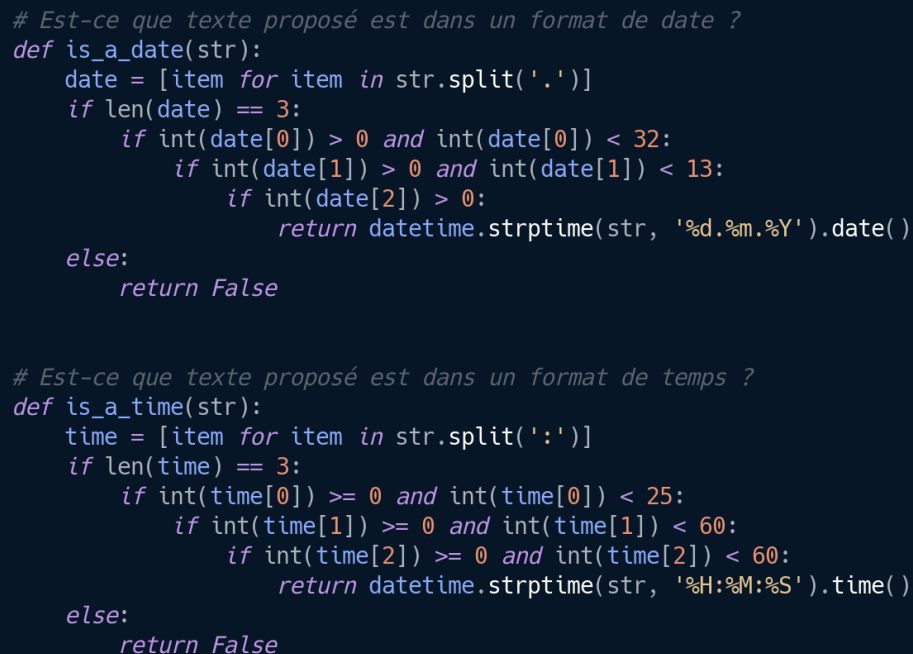
# Contrôle de présence de certains dossiers obligatoires
def check_directories(self):
    for host in [self.ftp_input, self.ftp_output]:
        if host.type == 'input':
            directories = self.get_directories('.', host)
            if 'Erreur' not in directories:
                self.create_directory('.', host, 'Erreur')
            if 'Fichier traités' not in directories:
                self.create_directory('.', host, 'Fichier traités')
```

Figure 17 : Capture d'écran des méthodes de contrôle et de récupération dans `server.py`

H. Utils.py

La page *utils.py* contient, quant à elle, plusieurs méthodes de vérification des types (boolean, Date, Time).

Ces méthodes sont supplémentaires à celles que nous utilisons. Elles nous permettent de contrôler plusieurs paramètres tels que la date et l'heure.



```
# Est-ce que texte proposé est dans un format de date ?
def is_a_date(str):
    date = [item for item in str.split('.')]
    if len(date) == 3:
        if int(date[0]) > 0 and int(date[0]) < 32:
            if int(date[1]) > 0 and int(date[1]) < 13:
                if int(date[2]) > 0:
                    return datetime.strptime(str, '%d.%m.%Y').date()
    else:
        return False

# Est-ce que texte proposé est dans un format de temps ?
def is_a_time(str):
    time = [item for item in str.split(':')]
    if len(time) == 3:
        if int(time[0]) >= 0 and int(time[0]) < 25:
            if int(time[1]) >= 0 and int(time[1]) < 60:
                if int(time[2]) >= 0 and int(time[2]) < 60:
                    return datetime.strptime(str, '%H:%M:%S').time()
    else:
        return False
```

Figure 18 : Capture d'écran des méthodes de vérification dans *utils.py*

IV. Pistes d'améliorations

Tout au long de la construction de notre code, nous avons rencontré plusieurs pistes d'amélioration que nous pourrions intégrer à notre script.

Tout d'abord, nous avons imaginé que le programme puisse reconnaître les erreurs d'annotation et, dans la mesure du possible, les corriger automatiquement. Par exemple en transformant une variable mal annotée (25,0) en une variable utilisable (25.0).

Lors de la notification à l'utilisateur, les corrections automatiques seraient alors mentionnées en plus et, dans le cas où aucune correction n'a été nécessaire, que les lignes ont été supprimées seulement.

Ensuite, nous pourrions penser que le programme ait la possibilité de reconnaître lorsque des colonnes sont conformes à nos *Header* attendus, mais qu'elles ne se situent pas à la bonne place. De ce fait il considérerait le fichier sans erreurs et qu'il replace les colonnes au bon endroit.

Pour terminer, nous trouverions idéal qu'en cas d'erreur de traitement de fichier, le programme envoie une requête par mail à l'utilisateur s'il est nécessaire de supprimer le fichier ou s'il doit être stocké dans un dossier à part. Ce mail contiendrait, dans le meilleur des cas, un hyperlien qui lancerait un script qui permette de faire l'une ou l'autre des solutions.

V. Conclusion

C'est un projet que tous les membres du groupe ont jugé difficile à réaliser. Un long travail pré-codage a dû être réalisée pour simplifier/synthétiser les processus de traitement/connexion qu'exécute notre script.

Nous avons ainsi dû, en pseudo-code, penser des objets, des méthodes et des variables et réaliser une structure complète pour démarrer le processus de codage avec confiance.

Nous avons sous-estimé le temps nécessaire à réaliser le travail de modélisation. La phase de codage du projet a néanmoins été rapidement clôturée.

Ce projet nécessiterait un plus large approfondissement de la gestion des erreurs.

Notre gestion de la connexion FTP ainsi que le traitement/contrôle global des fichier/dossiers est un point où notre groupe a apporté un soin particulier.

L'ensemble du groupe est content du travail transmis et confiant qu'il répond aux attentes du cahier des charges.

VI. Conclusions personnelles

A. Benoît Le Goff

Ce projet a nécessité beaucoup de travail pour s'informer et se former sur le langage python ainsi que ses bonnes pratiques. Néanmoins, notre travail préliminaire pour séparer le travail en classe python ainsi qu'en méthodes a permis de détacher une structure claire permettant de coder sereinement.

Les opérations de connexion aux FTP ainsi que la gestion des courriels ont été les plus simples.

A contrario le traitement des fichiers ainsi que leur transformation en objet utilisable python a été plus compliqué : J'ai démarré en utilisant la librairie csv qui ne m'a jamais donné satisfaction dans son utilisation selon notre cahier de charge du projet. J'ai switché sur la librairie pandas finalement pour traiter notre fichier.

C'est une librairie utilisée fréquemment en data science et que j'ai découvert via ce projet Cela m'a demandé quelques heures pour parcourir la large documentation et identifier les méthodes utiles pour notre projet.

Finalement, je suis content de notre travail de groupe et l'application de nos différentes phases de projet (modélisation, codage, test).

B. Tony de Carvalho

Ce projet a été d'une part très enrichissante pour moi, ayant des connaissances dans le code (PHP, Javascript), il a été intéressant de travailler dans un langage que je n'ai jamais pratiqué et dans un environnement (Linux) qui m'est encore très étranger. De ce fait une grande partie du temps de travail a été alloué à la recherche de solutions efficaces et simple pour accomplir ce projet ainsi qu'un apprentissage autodidacte de l'environnement de développement.

L'un des plus gros défis de ce projet a été de m'habituer à la syntaxe utilisée dans python, le manque d'accolades a fait qu'il m'était parfois difficile de me retrouver dans le code, je me suis aussi parfois perdu dans les variables dont le type n'a pas besoin d'être déclaré.

Sur une note un peu plus négative, le projet m'a paru poussée par rapport à la matière vu en cours pour une personne n'ayant pas d'expérience au préalable dans la programmation.

Finalement un bon travail de groupe en s'appuyant sur les forces de chaque membre nous a permis de finir ce projet de façon efficace.

C. Damjan Grujicic

Ayant quelques connaissances basiques de programmation, j'ai notamment pu suivre l'évolution du code au travers de ce projet.

Nous avons su tirer profit des forces de chacun, tout en ayant une approche d'entraide au sein de notre groupe. Nous avons su nous répartir le travail équitablement afin de garder notre objectif en tête qui était de réaliser un code de qualité et un rapport détaillé.

J'ai notamment pu approfondir mes connaissances sur un langage qui m'était peu familier.

D. Christophe Loureiro

En tant que membre de ce groupe avec peu d'expérience dans les langages de programmation, et en particulier Python, il m'a été difficile de savoir comment aborder cette thématique.

Néanmoins, grâce aux aides bienvenues de Benoît et de Tony, qui ont déjà de l'expérience dans l'univers du code et de la programmation, il m'a été tout de suite plus clair sur la manière de procéder.

Leurs explications m'ont permis de me familiariser avec les classes, notamment lorsque nous avons séparé le travail en classes python, ce qui m'a permis de mieux comprendre notre projet dans sa globalité.

Tout le monde a été mis à contribution, que ce soit dans l'approche technique que dans la rédaction du rapport.

De par mon peu d'expertise technique, j'ai beaucoup appris sur la méthode de travail en ayant notamment suivi de près la structure du code, en m'informant au préalable sur les prérequis ainsi que sur les *best practices* de python.

Annexe

Figure 1 :

Diagramme portant sur le Cheminement de fichiers.

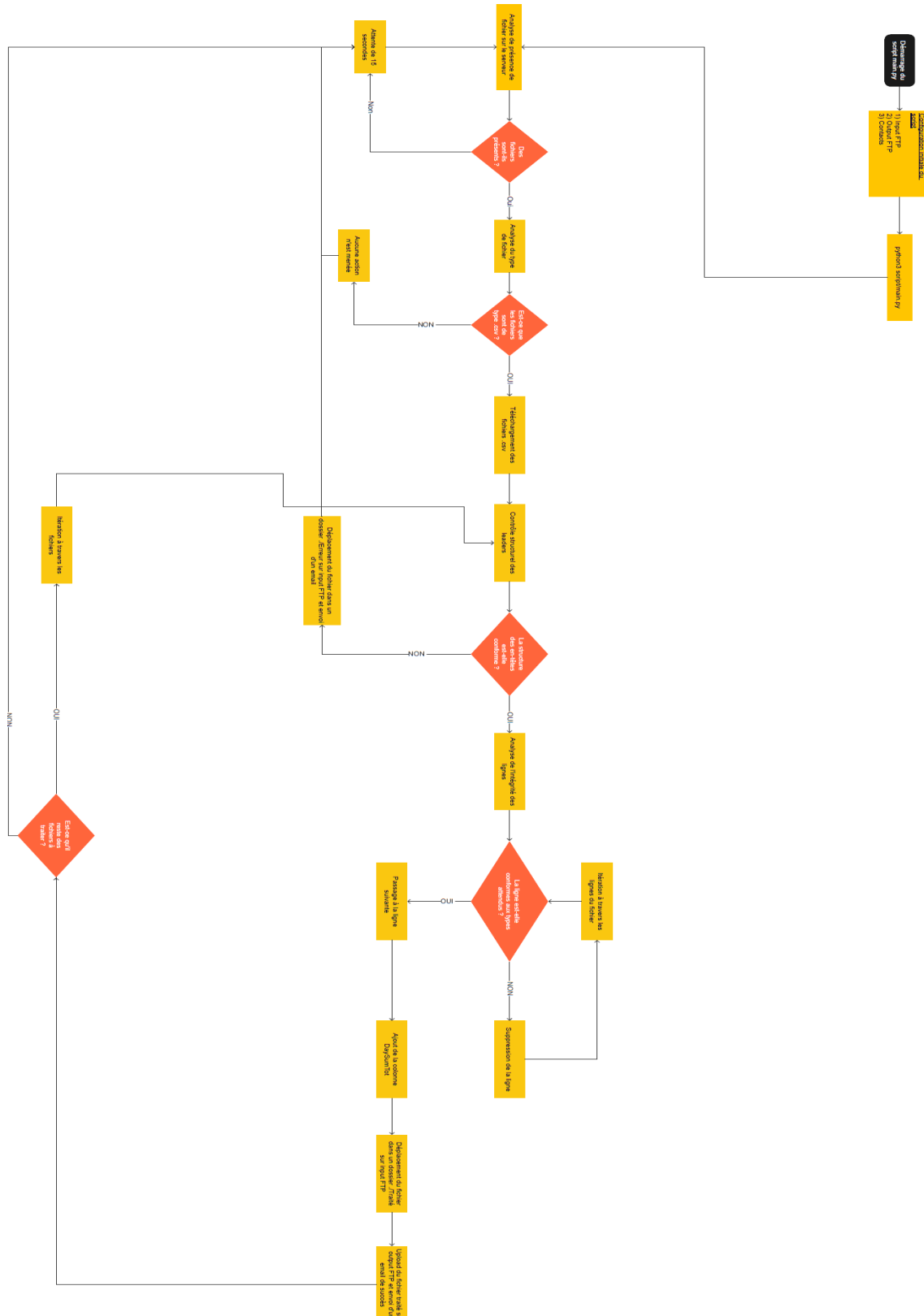


Figure 2 :

Diagramme de classes du projet python.

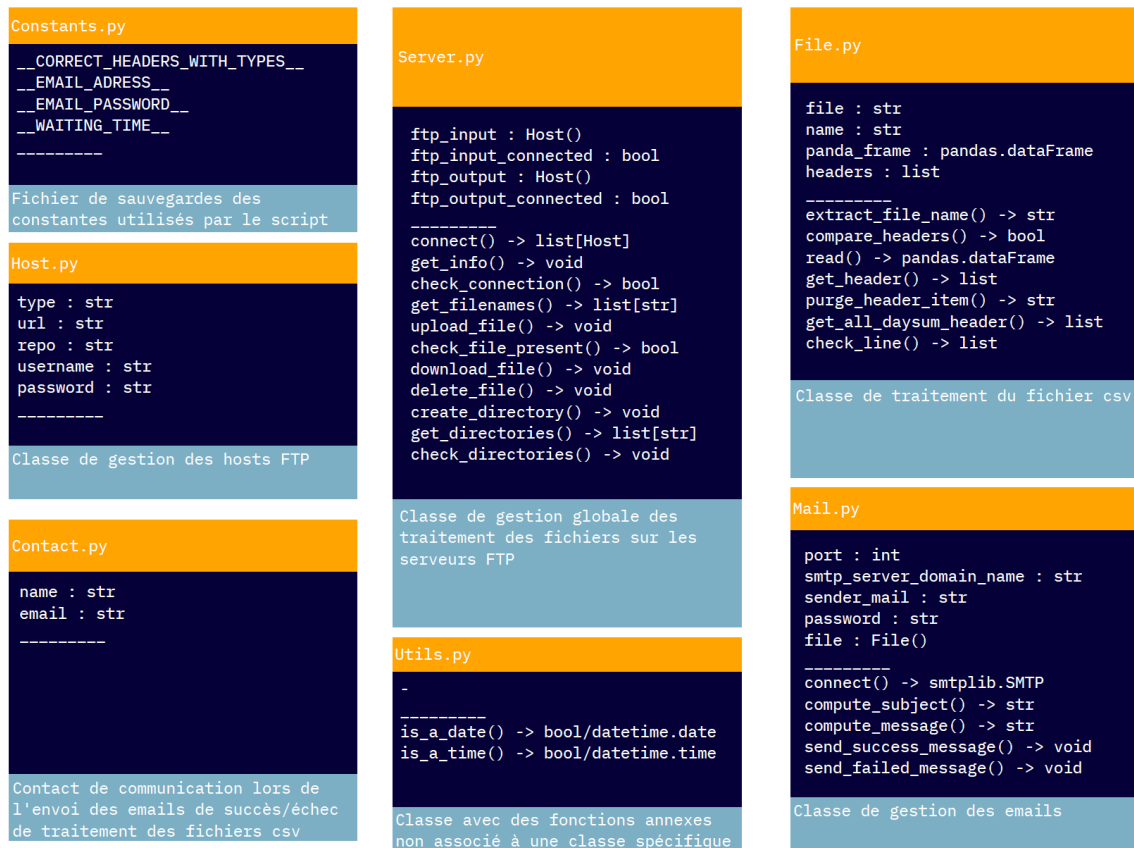


Table des abréviations

.csv	comma-separated values
FTP	File Transfert Protocol
HES-SO	Haute Ecole Spécialisée de Suisse Occidentale
ISO	International Organization for Standardization
Mo	Mégaoctets
OS	Operating System
.py	Extension d'un fichier du langage Python
RAM	Random Access Memory
url	Uniform Resource Locator

Table des illustrations

Figure 1: Site officiel de OpenSuse	2
Figure 2 : Création d'une VM dans le programme Oracle VM VirtualBox.....	3
Figure 3 : Configuration de la VM dans le programme Oracle VM VirtualBox.....	3
Figure 4 : Emplacements des fichiers de la VM via le programme Oracle VM VirtualBox	4
Figure 5 : Machine Virtuelle disponible dans notre programme Oracle VM VirtualBox	4
Figure 6 : Editer les propriétés de notre VM dans le programme Oracle VM VirtualBox	5
Figure 7 : Installation de Visual Studio Code dans notre environnement Linux.....	6
Figure 8 : Capture d'écran de constants.py.....	8
Figure 9 : capture d'écran class Contact de contact.py.....	9
Figure 10 : capture d'écran class File de file.py	9
Figure 11 : Capture d'écran de la méthode get_headers dans file.py	10
Figure 12 : Capture d'écran de la méthode compare_headers dans file.py	10
Figure 13 : Capture d'écran des méthodes dans mail.py	11
Figure 14 : Capture d'écran de méthodes dans main.py	12
Figure 15 : Capture d'écran de server.py.....	12
Figure 16 : Capture d'écran des méthodes de téléchargement et de suppression dans server.py	13
Figure 17 : Capture d'écran des méthodes de contrôle et de récupération dans server.py.....	14
Figure 18 : Capture d'écran des méthodes de vérification dans utils.py	15