Création d'un outil de « *reporting* » qui permet localiser les gros fichiers du disque

Fichiers à joindre : Creation_Onglets.py / Creation_Camembert.py / Creation_Legendes.py / Creation_Boutons.py

1. Objectifs :

   Il arrive que le disque dur d'un PC (Windows, MacOS ou Linux) ne dispose plus de beaucoup d'espace libre. Ce problème peut empêcher l'installation de nouvelles applications voire carrément empêcher la machine de démarrer ou de se mettre à jour…

   Pour gagner rapidement de l'espace libre, il suffit de supprimer quelques « gros » fichiers peu où plus utilisés mais toujours présents sur le disque dur. Ces fichiers sont souvent des vidéos, de grosses photos, des images d'installations ou encore des « sauvegardes » de clés USB.

   La difficulté est de localiser ces « gros » fichiers sur le disque durs.

   Dans le cadre de son activité, un administrateur système qui a en charge la maintenance d'un parc de PC (fixe ou portables) est souvent amené à effectuer ce type de maintenances.

   Le but de cette SAÉ est de développer un outil dit de « ***reporting*** » capable d'analyser tous les fichiers d'une arborescence qui se trouvent dans un répertoire donné (appelé **`répertoire_de_base`**) et de localiser les fichiers qui ont une taille supérieure à un seuil donné (1MiB / 10 MiB, etc…) tout en se limitant au 100 plus « gros » fichiers.

2. Contraintes :

   L'usage de cet outil de « ***reporting*** » doit être réservé aux utilisateurs avertis car il va permettre de supprimer des « gros » fichiers dont il faut connaître l'usage ou la fonction. La suppression de fichiers « systèmes » peut déstabiliser le fonctionnement du PC…

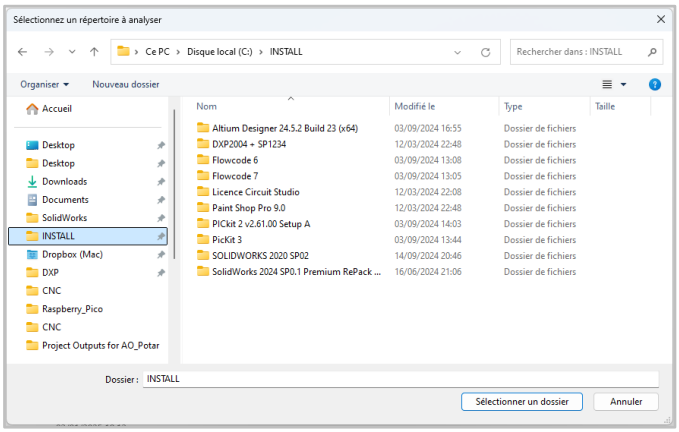
   De ce fait, le lancement de cet outil devra se faire par l'exécution d'un script « ***PowerShell*** ».

   Les étapes que doit effectuer ce script « ***PowerShell*** » sont :

   ➢ Permettre à l'utilisateur de sélectionner le « **`répertoire_de_base`** » avec ***un 1er script Python***.

   > ***Contraintes*** : *Cela doit se faire avec la méthode « **`getExistingDirectory(…)`** » des objets « **`QFileDialog`** » de la bibliothèque « **`PyQt5.QtWidgets`** »…*
   >
   > *Le script Python doit retourner au script « PowerShell » le répertoire sélectionné…*

   On doit avoir un résultat ressemblant à cela :



- Analyser avec ***un 2ème script Python***, toute l'arborescence du « **répertoire_de_base** ». Cela revient à faire l'inventaire de tous les fichiers du « **répertoire_de_base** » ainsi que tous ces sous-répertoires de manière récursive.

  A chaque fichier va être associé une liste :

  [*chemin_complet/nom_fichier, taille_du_fichier_en_octets*]

  L'analyse de l'arborescence du « **répertoire_de_base** » doit donc retourner une liste contenant autant de listes (comme décrites ci-dessus) qu'il y a de fichiers.

  Indications *: La manipulation des chemins de fichiers de Windows en Python avec des fichiers JSON pose un problème lié au symbole « \ » (antislash) qui sert de séparateur dans les chemins de fichiers mais qui représente également le caractère d'échappement en Python… Merci à Bill GATES et Paul ALLEN !!!*

  Pour mémoire, sur MacOS et Linux, le séparateur des chemins de fichiers est le symbole « / » (slash). Ce dernier ne pose pas de problème en Python et JSON…

  Les solutions Python pour régler ces problèmes sont traitées en détails dans un complément de cours.

  Il va ensuite falloir trier cette liste dans l'ordre des tailles de fichiers autrement dit placer en premier, la liste correspondant au plus gros fichier et en dernier celle du plus petit.

  Indications *: Vous pouvez utiliser la méthode « **sorted(…)** » associée aux listes en Python. La mise en œuvre du fonction anonyme « **lambda**… » peut être utile. A vous d'effectuer les recherches nécessaires.*

  Puis, on ne va retenir que les 100 (s'il y en a plus de 100) premières listes associées aux 100 plus gros fichiers…

  Indications *: Juste avec un « slicing »…*

  Pour finir, il faut stocker cette liste de liste dans un fichier JSON.

  Indications *: Vous pouvez utiliser la méthode « **dump(…)** » de la bibliothèque « **json** » (**import json**) pour réaliser cela… Là encore il va falloir prendre des précautions avec le symbole « \ » dans les chemins de fichiers…*

  Le résultat final doit être un fichier JSON ayant cette structure :

```
[
  ["C:\\INSTALL\\Juke Box\\Protel\\Jmv\\JUKEBOXO.DDB", 43134976],
  ["C:\\INSTALL\\library\\IPC Footprints\\Square IPC.ddb", 13035520],
  ["C:\\INSTALL\\Juke Box\\Protel\\Jmv\\TCDEJUKE.DDB", 9203712],
  ["C:\\INSTALL\\Juke Box\\Protel\\Jmv\\CALLID.DDB",7020544],
  ["C:\\INSTALL\\library\\Generic Footprints\\Advpcb.ddb", 5840896],
  ["C:\\INSTALL\\Juke Box\\Pic doc & pgm\\18C\\SHEET.PDF",5392979],
  ["C:\\INSTALL\\library\\IPC Footprints\\Rectangular IPC.ddb", 4540416],
  ["C:\\INSTALL\\library\\Generic Footprints\\Tapepak.ddb", 3108864],
  ["C:\\INSTALL\\library\\Connectors\\Samtec Connectors.ddb", 2617344]
]
```

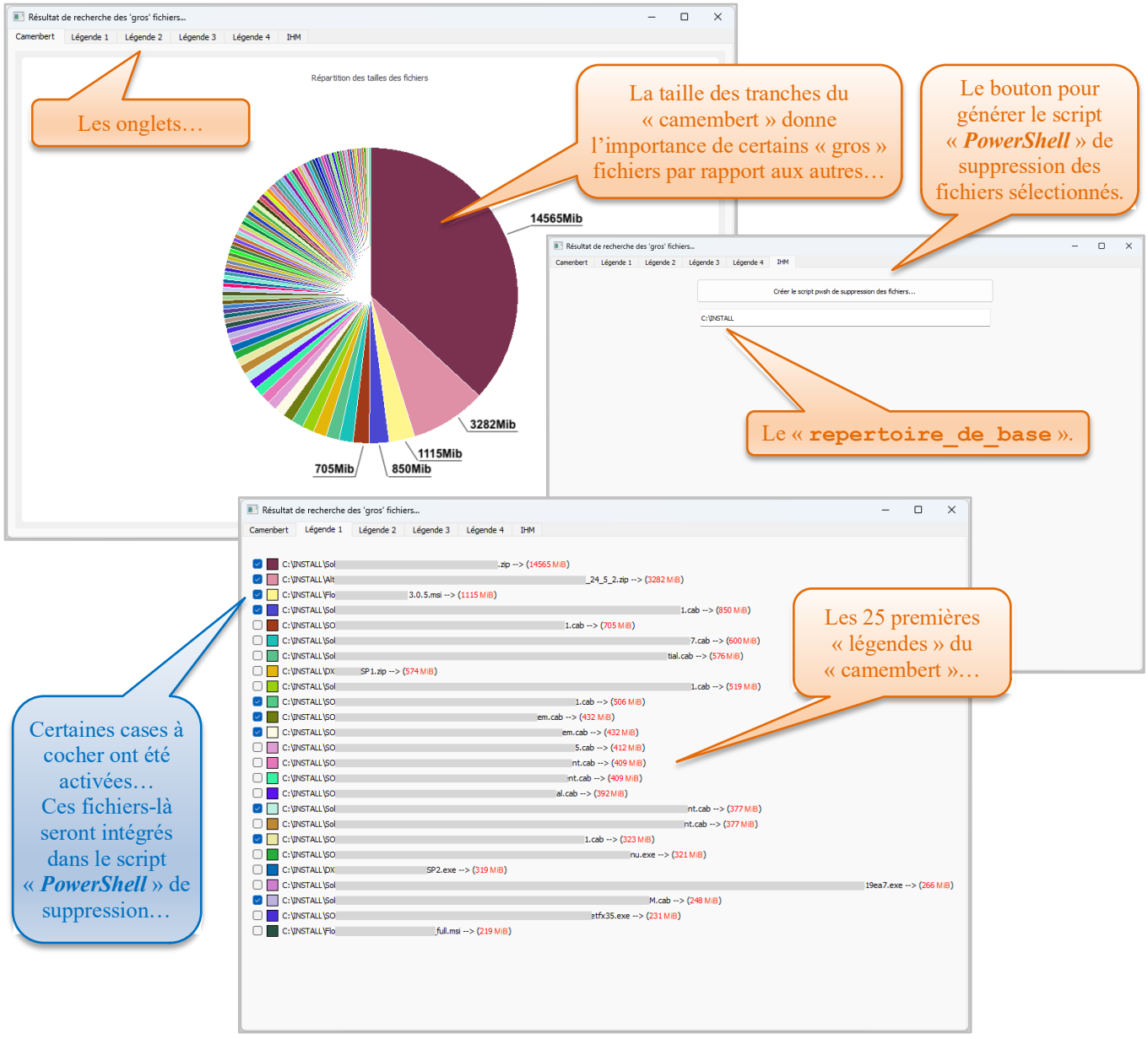
  Contraintes *: Le « **repertoire_de_base** » va devoir être passé en argument au script Python dans le script « **PowerShell** »…*

- Afficher avec ***un 3ème script Python***, le résultat de la recherche des « gros » fichiers dans toute l'arborescence du « **répertoire_de_base** ». Il est demandé d'effectuer un affichage graphique d'un « ***camembert*** » qui donne une indication relative de la taille des « gros » fichiers. Cette représentation doit être complétée par des listes de légendes. Chaque ligne de légende offre une case à cocher, un rappel de la couleur présente sur le « ***camembert*** » et le nom complet du fichier ainsi que sa taille en rouge. Comme le « ***camembert*** » peut contenir jusqu'à 100 portions, les 100 légendes ne peuvent pas s'afficher sur une seule page. Avec un affichage de 25 légendes par page, l'ensemble des légendes tiennent sur 4 pages.

  Pour afficher toutes ces pages, il est pratique de mettre en œuvre une fenêtre à onglets…

  On va de plus, ajouter un onglet supplémentaire pour offrir à l'utilisateur un bouton qui va permettre de générer **un script « *PowerShell* »** qui va permettre de supprimer tous les fichiers dont la case à cocher a été activée…

  Le résultat graphique est le suivant :



L'affichage de la fenêtre à onglets, du « ***camembert*** », des légendes et de l'onglet avec les boutons va se faire avec un code objet Python qui vous est donné. Des exemples de mise en œuvre de ces différents objets sont également donnés ci-dessous :

- La création des onglets se fait avec l'objet « **`Onglets`** » dans le script « `Creation_Onglets.py` » et dont le code est le suivant :

```python
from PyQt5.QtWidgets import QWidget, QTabWidget, QVBoxLayout, QMainWindow

class Onglets(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Résultat de recherche des 'gros' fichiers...")
        self.setGeometry(200, 100, 1000, 700)

        # Créer un QTabWidget
        self.onglets = QTabWidget()
        self.setCentralWidget(self.onglets)


    def add_onglet(self, titre, layout_a_ajouter):
        # Création d'un onglet...
        onglet = QWidget()
        layout_onglet = QVBoxLayout()
        layout_onglet.addWidget(layout_a_ajouter)
        onglet.setLayout(layout_onglet)

        # Ajout de l'onglet au QTabWidget
        self.onglets.addTab(onglet, titre)
```

Le « constructeur » hérite de toutes les propriétés et méthodes de l'objet « **`QMainWindow`** »…

Titre et position de la fenêtre (« **`QMainWindow`** »)…

On ajoute à la fenêtre un objet qui va intégrer des onglets (« **`QTabWidget`** »)…

La méthode « **`add_onglet(…)`** » va :

1. Créer un nouvel objet graphique (« **`QWidget`** »).
2. Créer un nouveau conteneur : « **`layout`** ».
3. Ajouter un contenu graphique à ce conteneur. Ce contenu (camembert, légende, …) est donné en argument à la méthode.
4. Affecter le conteneur à l'objet graphique (onglet).
5. Ajouter le nouvel onglet à l'objet qui intègre tous les onglets…

Exemple de mise en œuvre :

```python
from Creation_Onglets import Onglets
from PyQt5.QtWidgets import QApplication
import sys

NB_LEGENDES_PAR_PAGE = 25

if __name__ == "__main__":
    ......
    appli = QApplication(sys.argv)
    fenetre = Onglets()
    ......
    fenetre.add_onglet("Camembert", layout_fromage)
    liste_legende = []
    for num_page_legende in range(len(liste_fichiers) // NB_LEGENDES_PAR_PAGE):
        ......
        fenetre.add_onglet(f"Légende {num_page_legende + 1}", layout_legende)
    ......
    fenetre.add_onglet("IHM", layout_ihm)
    fenetre.show()
    sys.exit(appli.exec_())
```

Création de l'environnement graphique « **`PyQt`** ».

Création du conteneur d'onglets.

Ajout d'un onglet contenant le « camembert ».

Ajout de 1, 2, 3 ou 4 onglets contenant les légendes.

Ajout d'un onglet contenant le bouton de l'IHM.

Affichage de la fenêtre graphique…

- La création du camembert se fait avec l'objet « **`Camembert`** » dans le script « `Creation_Camembert.py` » et dont le code est le suivant :

```python
from PyQt5.QtChart import QChart, QChartView, QPieSeries
from PyQt5.QtGui import QFont

class Camembert:
    def __init__(self, liste_fichiers, liste_couleurs):
        self.liste_fichiers = liste_fichiers
        self.liste_couleurs = liste_couleurs

    def dessine_camembert(self):
        """
        Retourne une Widget Layout PyQt contenant un graphique circulaire type camembert.
        """
        if not self.liste_fichiers:
            raise ValueError(f"La liste doit contenir au moins 1 fichier.")

        # Création de la série de données pour le graphique
        series = QPieSeries()
        series.setLabelsVisible(True)
        taille_totale = 0
        for liste in self.liste_fichiers:
            taille_totale += liste[1]

        # Création des différentes tranches du camembert
        font = QFont("Arial Narrow", 12, QFont.Bold)
        for path_fichier, taille_fichier in self.liste_fichiers:
            etiquette = f"{taille_fichier // 1048576}MiB"
            pourcentage = taille_fichier / taille_totale * 100
            slice_ = series.append(etiquette, pourcentage)
            slice_.setBrush(self.liste_couleurs[len(series)-1])  # Couleurs dynamiques
            slice_.setLabelFont(font)
            slice_.setLabelPosition(slice_.LabelPosition.LabelOutside)

        # Affichage d'une étiquette pour les grandes tranches du camembert
        for slice_ in series.slices():
            slice_.setLabelVisible(slice_.angleSpan() > 6)

        # Création du graphique
        fromage = QChart()
        fromage.addSeries(series)
        fromage.setTitle("Répartition des tailles des fichiers")
        fromage.legend().hide()

        # Configuration du graphique avec QChartView
        layout_fromage = QChartView(fromage)

        return layout_fromage
```

Le « ***constructeur*** » admet 2 arguments :
- la liste des listes de fichiers & tailles.
- une liste de 100 couleurs aléatoires.

La méthode « **`dessine_camembert`** » va construire une page (« **`layout`** ») contenant tous les éléments graphiques du « ***camembert*** ».

Levée d'une erreur si la liste des fichiers est vide…

Création d'une série de données pour un objet de type « ***camembert*** ».

Calcul de la taille totale des tous les fichiers représentés dans le camembert. Ce total sera nécessaire pour attribuer une dimension à chaque tranche du « ***camembert*** »…

Création de chaque tranche du « ***camembert*** » avec définition de son étiquette (position, police et texte), de sa taille (secteur angulaire), et de sa couleur…

Création de l'objet graphique de type « ***camembert*** ».

Ajout de la série de données à l'objet graphique.

Définition du titre du « ***camembert*** ».

On cache les légendes du « ***camembert*** ».

Création du conteneur (« **`layout_fromage`** ») qui contient tous les éléments graphiques du « ***camembert*** »…

La méthode « **`dessine_camembert`** » retourne ce conteneur graphique pour être intégrée à un onglet lors de sa création…

Exemple de mise en œuvre :

```
from Creation_Camembert import Camembert

if __name__ == "__main__":
    ......
    fromage = Camembert(liste_fichiers, liste_couleurs)
    layout_fromage = fromage.dessine_camembert()
    fenetre.add_onglet("Camembert", layout_fromage)
    ......
```

Instanciation et initialisation de l'objet « **Camembert** ».

Construction graphique du conteneur (« **layout_fromage** ») par appel de la méthode « **dessine_camembert()** ».

- La création des légendes se fait avec l'objet « **Legendes** » dans le script « `Creation_Legendes.py` » et dont le code est donné ci-dessous.

> ***Note*** *: Comme on doit être capable de traiter entre 1 et 100 « gros » fichiers, les légendes doivent se répartir sur 1 à 4 pages. Pour faire cette répartition, on peut utiliser un index qui va valoir 0, 25, 50, ou 75 et qui indique quelles sont les légendes à afficher sur la page parmi les éléments de la liste des fichiers…*

```
from PyQt5.QtWidgets import QWidget, QVBoxLayout,  QLabel, QHBoxLayout, QCheckBox
from PyQt5.QtCore import Qt

class Legendes:
    def __init__(self, liste_fichiers, liste_couleurs, num_legende_start):
        self.liste_fichiers = liste_fichiers
        self.liste_couleurs = liste_couleurs
        self.num_legende_start = num_legende_start
        self.num_legende_stop = min(len(liste_fichiers), num_legende_start+25)
        self.cases_a_cocher = []

    def dessine_legendes(self):
        """
        Retourne une Widget Layout PyQt contenant une liste de légendes.
        """
        legendes = QWidget()
        layout_legendes = QVBoxLayout(legendes)
        layout_legendes.setContentsMargins(0, 20, 0, 20)
        layout_legendes.setAlignment(Qt.AlignTop)

        # Ajouter des datas à chaque légende
        for num_slice in range(self.num_legende_start, self.num_legende_stop):
            # Création de la case à cocher
            case_a_cocher = QCheckBox()
            case_a_cocher.setChecked(False)  # Définir l'état initial de la case (cochée ou non)
            self.cases_a_cocher.append(case_a_cocher)


            # Création d'un petit carré coloré
            rectangle_colore = QWidget()
            rectangle_colore.setFixedSize(16, 16)
            couleur = self.liste_couleurs[num_slice].name()
            rectangle_colore.setStyleSheet(f"background-color: {couleur}; border: 1px solid black;")
```

Le « ***constructeur*** » admet 3 arguments :
- la liste des listes de fichiers & tailles.
- la liste de 100 couleurs aléatoires.
- un index indiquant les 25 légendes à considérer pour l'affichage.

La méthode « **dessine_legendes** » va construire une page (« **layout** ») contenant tous les éléments graphiques des 25 légendes.

Création d'un objet graphique (« **QWidget** »).

Création d'un nouveau conteneur : « **layout** ».

Paramétrage de marges et d'alignement…

Pour chacun des 25 légendes :

Création d'une case à cocher (« **QCheckBox** »). On ajoute cette case dans une liste pour faciliter une lecture ultérieure des états de toutes ces cases…

Ajout d'un petit rectangle (« **QWidget** ») de couleur identique à celle de la tranche du « ***camembert*** ».

Dimensions du rectangle (carré en fait).

Paramétrage (dimensions et couleur) du rectangle.
A noter : Cela se fait avec une syntaxe CSS !!!

```
            # composition du contenu textuel de la légende
            chemin_fichier = self.liste_fichiers[num_slice][0]  # Extrait le chemin et nom du fichier
            taille_fichier = self.liste_fichiers[num_slice][1] // 1048576  # Extrait la taille du fichier
            etiquette_legende = f"<span style='color:black;font-family:Arial Narrow'>{chemin_fichier} -->
                    (</span><span style='color:red;'>{taille_fichier} MiB</span>)"
            texte_legende = QLabel(etiquette_legende)

            # Disposition horizontale pour l'élément de légende
            ligne_legende = QHBoxLayout()
            ligne_legende.setAlignment(Qt.AlignLeft)
            ligne_legende.setContentsMargins(5, 0, 5, 5)
            ligne_legende.stretch(2)
            ligne_legende.addWidget(case_a_cocher)
            ligne_legende.addWidget(rectangle_colore)
            ligne_legende.addWidget(texte_legende)


            legende = QWidget()
            legende.setLayout(ligne_legende)
            legende.setContentsMargins(0, 0, 0, 0)

            layout_legendes.addWidget(legende)
            layout_legendes.setSpacing(0)

        return legendes

    def recupere_etats_case_a_cocher(self):
        etats = []
        for case in self.cases_a_cocher:
            etats.append(case.isChecked())
        return etats
```

Composition du contenu de la légende (nom complet du fichier et taille). Le paramétrage graphique se fait également avec la syntaxe CSS ! Puis création de l'étiquette (« **QLabel** »).

Création d'un conteneur (« **layout** ») pour 1 légende.

Paramétrage de ce conteneur.

Ajout de la case à cocher, du rectangle de couleur et du texte à ce conteneur.

Création d'un objet « **QWidget** » pour une ligne de légende.

Création d'un conteneur : « **layout** ».

Ajout au conteneur des lignes de légendes, le conteneur d'une ligne de légende…

La méthode « **dessine_legendes** » retourne ce conteneur graphique pour être intégrée à un onglet lors de sa création…

La méthode « **recupere_etats_case_a_cocher** » établit une liste de booléen indiquant les états des cases à cocher de la page de légendes sélectionnée…

Exemple de mise en œuvre :

```
from Creation_Legendes import Legendes

NB_LEGENDES_PAR_PAGE = 25

if __name__ == "__main__":
    ......
    liste_legende = []
    for num_page_leg in range(len(liste_fichiers) // NB_LEGENDES_PAR_PAGE):
        liste_legende.append(Legendes(liste_fichiers, liste_couleurs, NB_LEGENDES_PAR_PAGE * num_page_leg))
        layout_legende = liste_legende[num_page_leg].dessine_legendes()
        fenetre.add_onglet(f"Légende {num_page_leg + 1}", layout_legende)
    ......
```

Instanciation et initialisation de l'objet « **Legendes** ».

Construction graphique du conteneur (« **layout_legende** ») par appel de la méthode « **dessine_legendes()** ».

- La création du contenu de l'onglet de bouton se fait avec l'objet « **Boutons** » dans le script « `Creation_Boutons.py` » et dont le code est donné ci-dessous.

```
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QLineEdit
from PyQt5.QtCore import Qt

class Boutons:
    def __init__(self, repertoire_base, callback):
        self.repertoire_base = repertoire_base
        self.callback = callback

    def dessine_boutons(self):
        boutons = QWidget()
        layout_boutons = QVBoxLayout(boutons)
        layout_boutons.setSpacing(10)
        layout_boutons.setAlignment(Qt.AlignTop)

        bouton_genere_delete_script = QPushButton("Créer le script pwsh de suppression des fichiers...")
        bouton_genere_delete_script.setFixedSize(500, 40)
        bouton_genere_delete_script.clicked.connect(self.callback)

        layout_boutons.addWidget(bouton_genere_delete_script, alignment=Qt.AlignHCenter)

        text_repertoire_base = QLineEdit()
        text_repertoire_base.setText(self.repertoire_base)

        text_repertoire_base.setFixedSize(490, 30)
        layout_boutons.addWidget(text_repertoire_base, alignment=Qt.AlignHCenter)

        return boutons
```

Le « ***constructeur*** » admet 2 arguments :
- le « **repertoire_de_base** ».
- le nom de la fonction (appelée aussi « callback ») qui va traiter le clic sur le bouton…

Création d'un objet graphique (« **QWidget** »).

Création d'un conteneur : « **layout** ».

Paramétrage de marges et d'alignement…

Création d'un bouton (« **QPushButton** »).

Association de la « **callback** » à l'évènement « **click** »

Ajout du bouton au conteneur principal.

Création d'une zone de texte (« **QLineEdit** »).

Placement du « **repertoire_de_base** » dans le texte de cet objet.

Ajout de la zone de texte au conteneur principal.

La méthode « **dessine_boutons** » retourne ce conteneur graphique pour être intégrée à un onglet lors de sa création…

Exemple de mise en œuvre :

```
from Creation_Boutons import Boutons

if __name__ == "__main__":
    ......
    ihm = Boutons(repertoire_base, creation_script_suppression)
    layout_ihm = ihm.dessine_boutons()
    fenetre.add_onglet("IHM", layout_ihm)
    ......
```

Instanciation et initialisation de l'objet « **Boutons** ».

Le second argument est le nom de la fonction (« callback ») qui va être appelée lors du click sur le bouton…

Construction graphique du conteneur (« **layout_ihm** ») par appel de la méthode « **dessine_boutons()** ».

3. Travail demandé :

Codage du 2ème script Python (on fera le 1er par la suite)

– Codez sous Pycharm, une fonction Python qui va, à partir d'un « **repertoire_de_base** » construire une liste d'éléments où chaque élément est une liste contenant le nom complet (chemin absolu + nom) d'un fichier et sa taille en octets. Il est nécessaire d'avoir une telle liste pour chaque fichier présent dans le « **repertoire_de_base** » et tous ses sous-répertoires, sous-sous-répertoires, etc…

> ***Indications*** *: Utilisez les notions abordées lors du CM de cette SAÉ (bibliothèque « **pathlib** »)*

Cette fonction retourne la liste demandée.

- Codez le programme principal (main) qui permet de tester cette fonction. A ce stade de l'activité, le « **repertoire_de_base** » est figé et codé dans une variable du script Python.

- Codez une seconde fonction qui va trier **de manière décroissante** une liste de listes en utilisant comme clé, le second élément des listes de la liste principale. Cette fonction admet comme argument la liste de listes précédente et retourne la même liste mais triée.

> ***Indications*** *: Posez la question suivante à votre meilleur « ami » :*
>
> **« Donne-moi un cours sur la fonction lambda en python »**

- Codez une troisième fonction conserver uniquement les listes de la liste principale (donnée en arguments) dont le second élément, c'est-à-dire la taille en octets, est supérieur à une valeur « **TAILLE_MINI_FICHIER_EN_MEBI_OCTET** » passée en argument. Il faut de plus ne pas dépasser un nombre de fichiers également donné en argument : « **NB_MAXI_FICHIERS** ».

> ***Indications*** *: 1 MiB = 1024 kiB = 1024 x 1024 octets = 1048576 octets*

La fonction retourne le liste « filtrée ».

- Codez une quatrième fonction qui va créer un fichier JSON contenant la liste de listes passée en argument. Le nom du fichier JSON est également passé en argument.

> ***ATTENTION*** *: Pour placer dans un fichier JSON une chaîne de caractères contenant des « \ » (antislash), symbole de séparation des chemins d'arborescence sous Windows, il faut les remplacer par un double antislash (« \\ ») car en Python comme en JSON l'antislash est un caractère d'échappement…*
>
> *Le code suivant va être nécessaire :*

```
for liste_fichier in liste_fichiers:
    liste_fichier[0].replace('\\', '\\\\')
```

Codage du script « PowerShell »

- Si « ***PowerShell*** » existe sur votre PC, il se lance à partir d'un Terminal Windows (cmd) ou bien à partir du Terminal intégré à Pycharm () avec la commande « **`pwsh`** ». Si « ***PowerShell*** » n'est pas présent, installez-le…
- Lorsque vous êtes dans l'interface « ***PowerShell*** », l'invite de commande débute par « **`PS`** » :

```
PS /Users/olivier/Documents/IUT/PyCharm_Projects/Analyse_File_Tree>
```

- Dans l'interface « ***PowerShell*** », déplacez-vous dans le répertoire où est sauvegardé vos fichiers « `.py` »… (**`CD..`** ou **`CD`**)
- Codez directement sous Pycharm un script « ***PowerShell*** » qui lance le précédent script Python.

> ***Indications*** *: Votre script doit avoir pour suffixe « ps1 » pour que tout le monde puisse reconnaître un fichier « **PowerShell** » (exemple : «* `Analyse_Gros_fichiers.ps1` *»)*

- Testez votre script « ***PowerShell*** ».
- Modifiez vos codes Python et « ***PowerShell*** » pour que le « **`repertoire_de_base`** » soit passé en paramètre au script Python dans le script « ***PowerShell*** ».

Codage du 1er script Python

- Installez les bibliothèques nécessaires à la suite de cette activité avec les lignes de commandes depuis un terminal :

```
pip install PyQt5
pip install PyQtChart
```

  Vous pouvez aussi installez ces bibliothèques dans le gestionnaire « ***Python Package*** » de Pycharm ()…

- Codez dans nouveau script Python une fonction qui ouvre un sélecteur de répertoire « **`QFileDialog`** » de la bibliothèque « **`PyQt5.QtWidgets`** ».

  Le programme principale (**`main`**) de ce code doit retourner le répertoire sélectionné

> ***Indications*** *: Pour que votre code compatible multi-plateforme (Windows, MacOS, Linux), il convient d'utiliser la bibliothèque «* **`pathlib`** *»…*

Modification du script « *PowerShell* »

- Modifiez votre script « ***PowerShell*** » pour qu'il appelle le script Python que vous venez d'écrire, récupère le « **`repertoire_de_base`** » sélectionné et passe en paramètre ce répertoire au script Python codé au début de l'activité.

> ***Indications*** *: Vous pouvez vérifier dans le script «* ***PowerShell*** *» que la variable contant le «* **`repertoire_de_base`** *» n'est pas vide :* **`if (-not $rep_base) {…}`** *et que ce répertoire est valide :* **`if (Test-Path $rep_base) {…}`**

Codage du 3ème script Python

- Codez dans nouveau script Python une fonction qui récupère le contenu du fichier JSON. Le nom de ce fichier est donné en argument à la fonction et elle retourne une liste de listes qui contient comme précédemment les données (nom complet et taille) des plus « gros » fichiers.
- Codez une fonction qui génère et retourne une liste de **`NB_MAXI_FICHIERS`** couleurs aléatoires (format « **`QColor`** »).

> ***Indications*** *: Vous pouvez ajouter à la liste de couleur un objet :*
>
> `QColor(val_aleatoire_rouge, val_aleatoire_verte, val_aleatoire_bleu)`
>
> *La valeur de couleur de cette liste est directement utilisable dans une méthode « **`setBrush(…)`** ». Par contre, si vous souhaitez avoir l'expression hexadécimale de la couleur (exemple : **`#4F1A94`**), il faut utiliser la méthode « **`name()`** » (exemple : « **`couleur_hexa = liste_couleurs[i].name()`** »)…*

- Codez ensuite le programme principal (**`main`**) qui génère l'application graphique en suivant les étapes suivantes :
  - Le « **`repertoire_de_base`** » est pour l'instant donné par une variable.
  - Lecture du fichier JSON.
  - Création de la liste des couleurs aléatoires.
  - Création d'une fenêtre graphique avec onglets.
  - Création du conteneur (« **`layout`** ») du « ***camembert*** » et ajout de ce dernier à un onglet.
  - Création des 1,2,3 ou 4 conteneurs (« **`layout`** ») de légendes et ajouts des onglets nécessaires.
  - Création du conteneur (« **`layout`** ») de l'IHM (avec le bouton) et ajout de l'onglet correspondant.

> ***Indications*** *: Lors de l'instanciation de ce dernier « **`layout`** », vous devez choisir le nom de la fonction (« **`callback`** ») qui sera appelée lors du clic sur le bouton.*
>
> *Cette fonction doit forcément exister et peut pour l'instant ne rien contenir (enfin juste l'instruction « **`pass`** »).*

– Complétez le code de la fonction associée au click du bouton (« **`callback`** ») de manière à générer un nouveau script « ***PowerShell*** » ayant la structure donnée ci-dessous :

Les noms complets doivent être compatible à toutes les plateformes : Windows, MacOS ou Linux….

Il doit y avoir autant de ligne « **`Remove-Item -Path "……" –Force`** » que de case à cocher qui ont été activées…

> ***Indications*** *: Pour chaque page de légendes, vous pouvez appeler la méthode « **`recupere_etats_cases_a_cocher()`** »*

Sous Windows :

```
Write-Output "Script PowerShell pour supprimer des fichiers sans confirmation"
Write-Output "Attention : cette suppression est définitivement ..."

$reponse = Read-Host "Veuillez confirmer la suppression de tous ces fichiers : (OUI)"
if ($reponse -eq "OUI") {
    $confirmation = Read-Host "Etes-vous bien certain(e) ? (OUI)"
    if ($confirmation -eq "OUI") {
        Remove-Item -Path "C:\...nom_complet_fichier1" -Force
        Remove-Item -Path "C:\...nom_complet_fichier2" -Force
        Remove-Item -Path "C:\...nom_complet_fichier3" -Force
    } else {
        Write-Output "Opération annulée..."
    }
} else {
    Write-Output "Opération annulée..."
}
```

Sous MacOS ou Linux :

```
Write-Output "Script PowerShell pour supprimer des fichiers sans confirmation"
Write-Output "Attention : cette suppression est définitivement ..."

$reponse = Read-Host "Veuillez confirmer la suppression de tous ces fichiers : (OUI)"
if ($reponse -eq "OUI") {
    $confirmation = Read-Host "Etes-vous bien certain(e) ? (OUI)"
    if ($confirmation -eq "OUI") {
        Remove-Item -Path "/users/...nom_complet_fichier1" -Force
        Remove-Item -Path "/users/…nom_complet_fichier2" -Force
        Remove-Item -Path "/users/...nom_complet_fichier3" -Force
    } else {
        Write-Output "Opération annulée..."
    }
} else {
    Write-Output "Opération annulée..."
}
```

Complément de cours…

1. LA GESTION MULTIPLATEFORME DES CHEMINS & FICHIERS

Lorsque l'on code un script Python qui manipule des fichiers et les chemins correspondants, il est nécessaire de prendre quelques précautions notamment si ce code doit être interprété sous Windows…

Historiquement, « Microsoft » a fait le choix dans les premières versions du DOS, d'utiliser le symbole « \ » (antislash) comme séparateur de répertoire dans un chemin de fichier.

Exemple : « `C:\users\bill_gates\mauvaises_idees\chemins_avec\antislash.txt` »

Bien évidemment, tout comme la conduite à gauche outre-manche, il n'est pas possible de revenir en arrière !!!

Cela pose un problème non négligeable avec les langages (comme Python) qui utilise ce même symbole d'antislash comme caractère d'échappement dans les chaînes de caractères.

Note : ce problème n'existe pas sous MacOS et Linux où le séparateur de répertoire dans un chemin de fichier est le « / » (slash) !!!

Exemple sur MacOS : « `/users/steve_jobs/bonnes_idees/chemins_avec/slash.txt` »

2. LA SOLUTION PYTHON : LA BIBLIOTHEQUE « `pathlib` »

Python propose une bibliothèque qui va remédier aux problèmes cités ci-dessus. Elle s'importe avec :

```
from pathlib import Path
```

L'idée consiste d'associer ensuite un objet de type « **`Path`** » à toute chaîne de caractères qui représente un nom de fichier (relatif ou absolu), un chemin dans une arborescence d'OS…

Tout le travail qui doit s'opérer sur les noms de chemins et de fichiers se fait ensuite sur ces objets « **`Path`** ».

Lorsqu'un affichage sous forme de texte d'un chemin ou/et d'un nom de fichier doit se faire, un « *casting* » (conversion) en **`string`** sera nécessaire. Et suivant l'OS sur lequel est interprété le code Python, l'affichage se fera automatiquement au bon format…

Note : C'est magique…

```
import platform
from pathlib import Path

rep_perso = Path.home()        #répertoire personnel de l'utilisateur
print(str(rep_perso))
print(platform.system())
```

```
/Users/steve_jobs
Darwin
```

Surnom de MacOS…

```
C:\Users\bill_gates
Windows
```

3. LES AUTRES OUTILS DE LA BIBLIOTHEQUE « `pathlib` »

La bibliothèque « **`pathlib`** » offre de nombreuses autres méthodes et propriétés bien pratiques.

Dans un premier temps, il faut créer un objet « **`Path`** » à partir d'une chaîne de caractères qui contient un nom de fichier avec un chemin relatif ou absolu. Le format « / » ou « \ » importe peu…

La création de l'objet « **`Path`** » peut aussi être ce que retourne un « sélecteur de fichier »…

```
from pathlib import Path

fichier = "nom_fichier.txt"          # Chaîne de caractère du nom du fichier…
f_path = Path(fichier)               # Objet Path associé à ce fichier
```

Méthode	Description	Exemples

`resolve()`	Retourne un objet « Path » où le chemin complet a été ajouté au nom du fichier si ce n'était pas déjà le cas…	`f_path = f_path.resolve()`
		`--> /users/steve_jobs/nom_fichier.txt`
		`--> C:\users\bill_gates\nom_fichier.txt`

`name`	Nom du fichier.	`f_path.name --> nom_fichier.txt`
`suffix`	Suffixe du fichier.	`f_path.suffixe --> .txt`

`parent`	Chemin absolu du fichier.	`f_path = f_path.resolve()`
		`f_path.parent`
		`--> /users/steve_jobs`
		`--> C:\users\bill_gates`

`exists()`	Retourne un booléen indiquant si le fichier ou dossier existe ?	`f_path.exist() --> True`
`is_file()`	Retourne un booléen indiquant si c'est un fichier ?	`f_path.is_file() --> True`
`is_dir()`	Retourne un booléen indiquant si c'est un dossier ?	`f_path.is_dir() --> False`

Lorsque l'objet « **Path** » est un dossier, il est possible de le parcourir et de générer une liste contenant un objet « **Path** » pour chaque élément (dossier ou fichier) qui figure dans le dossier de départ… Cela permet de faire l'inventaire du contenu d'un dossier.

Méthode	Description	Exemples
`iterdir()`	Retourne une liste de tous les fichiers que contient le répertoire de départ…	`liste = f_path.iterdir()`

`glob("*.txt")`	Retourne une liste de tous les fichiers que qui vérifient une condition dans le répertoire de départ…	`liste_txt = f_path.glob("*.txt")`
		`liste_py = f_path.glob("*.py")`
		`liste_all = f_path.glob("*")`

`rglob("*")`	Méthode similaire à la précédente mais avec un parcours « récursif » de toute l'arborescence à partir du répertoire de départ (sous-répertoire, sous-sous-répertoire, etc…)	`liste_txt = f_path.rglob("*.txt")`
		`liste_py = f_path.rglob("*.py")`
		`liste_all = f_path.rglob("*")`

Exemples :

```
from pathlib import Path

# Exemple : Recherche récursive de tous les fichiers et dossier .txt
for file in Path("documents").rglob("*.txt"):
    print(file)
```

```python
from pathlib import Path

# Exemple : Recherche récursive de tous les fichiers uniquement (pas les dossiers)
for file in Path("documents").rglob("*"):
    if file.is_file():
        print(file)
```

```python
from pathlib import Path

# Exemple : Recherche récursive de tous les dossiers uniquement (pas les fichiers)
for file in Path("documents").rglob("*"):
    if file.is_dir():
        print(file)
```

```python
from pathlib import Path

# Exemple : Recherche récursive de 2 types de fichiers ou dossiers (.txt ou .pdf)
for file in Path("documents").rglob("*"):
    if file.suffix in [".txt", ".pdf"]:
        print(file)
```

```python
from pathlib import Path

# Exemple : Trouver les fichiers ou dossiers de plus de 1Mo...
for file in Path("documents").rglob("a*"):
    if file.stat().st_size > 1048576 :
        print(f'{file} - Taille : {file.stat().st_size / 1048576 :.2f} Mo')
```

```python
from pathlib import Path

# Exemple : Affichage du chemin absolu des fichiers ou répertoire .txt
for file in Path("documents").rglob("*.txt"):
    print(file.resolve())
```