27/09/2021

Python pour l'ingénieur

Exercice : Meteo

Module de formation

Introduction

- Le cours propose plusieurs exercices permettant de comprendre le fonctionnement et le paradigme propre du langage Python.
- Ce cahier propose un exercice d'un autre genre : suite d'étapes permettant de créer un programme complet, du moteur jusqu'à l'IHM graphique.
- Cet exercice complet est donc très proche du projet personnel réalisé en binôme.
- Thème : Manipulation et analyse de relevés météo

Base de données

- L'exercice s'appuie sur la base de données SYNOP de relevés météo fournie par Météo France installée sur **S:\PAI\database**
  - Source : https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=90&id_rubrique=32
- Elle comprend des relevés dans 62 villes de France, de 1996 à 2019, à intervalle régulier de 3 heures (~70000 relevés par ville).

- La base de données contient 1 fichier CSV par ville (le fichier a comme nom le code SYNOP de la ville)

  *Un fichier **CSV** est un fichier texte comprenant une ligne par relevé, et des colonnes séparées par un point-virgule (ou un autre séparateur). La première ligne est un 'header' qui donne le nom (champ) de chaque colonne. Un fichier CSV peut être ouvert et lu avec Notepad++ ou Excel.*
- La signification de chaque champ du fichier CSV est fournie par le fichier PDF doc_parametres_synop_168.pdf situé à la racine de la base de données.

- La liste des 62 villes est fournie par le fichier JSON postesSynop.json situé à la racine de la base de données. Ce fichier fournit le code international (5 chiffres), le nom et la position géographique de chaque ville.

  *Le format **JSON** est un format de données texte permettant de représenter de l'information structurée. Un fichier JSON peut être ouvert et lu avec Notepad++.*

Modules

- Créer les fichiers suivants :
  - `meteo1.py` : moteur du programme
  - `meteo1t.py` : interface utilisateur en mode texte
  - `meteo1g.py` : interface utilisateur en mode graphique

- Ces fichiers seront mis à jour à chaque étape. Pour l'étape n°2 par exemple, il sera nécessaire de copier et de renommer ces 3 fichiers, afin de travailler avec meteo2.py, meteo2t.py et meteo2g.py.

- Les fichiers meteo*g.py sont mis de côté en attendant le début du cours sur les interfaces graphiques.

- Chaque module développé doit contenir des tests qui doivent être codés au fur et à mesure du développement. Ces tests sont placés dans une (ou plusieurs) fonction test() en bas du module, appelée lors de l'exécution du module par les lignes :

```
if __name__ == '__main__':
    test()
```

Etape 1

- Objectif de l'étape 1 : Créer une application affichant les mesures météo d'une ville pour une journée donnée.
- Module meteo1.py
  - Définir une classe nommé **Measure** comprenant les 3 attributs suivants : date, temperature, rain. Pour représenter une date on utilisera la classe datetime du package datetime.
  - Dans un fichier SYNOP, une date est représentée par une chaine de caractères de format YYYYMMDDHHMMSS. Définir une fonction **str2date**(datestr: str, format: str) qui convertit une date sous forme de chaine de caractères en un objet datetime, qu'elle retourne. Utiliser la fonction strptime() du module datetime. Retourner None si la date est invalide (utiliser un bloc try/except).
  - Définir une fonction **readsynopfile**(filename: str) qui lit dans le fichier csv filename les données d'une ville et les retourne dans une liste.
    - Implémenter dans cette fonction la lecture du fichier csv fourni en argument (filename) en utilisant les fonctions open(), readline(), split(), strip().
    - La fonction retourne une liste contenant les mesures de la ville (liste d'objets de la classe Measure).
    - Noter que certaines données peuvent être manquantes : elles sont marquées 'mq' dans le fichier SYNOP. Dans ce cas considérer une température = 0 K et une précipitation = 0 mm.
    - Utiliser le fichier doc_parametres_synop_168.pdf et éditer un fichier CSV avec Excel pour repérer l'index des colonnes utiles à l'exercice (code de ville, date, température, précipitations sur les 3 dernières heures). Le module utilise ces index en dur.
  - Définir une fonction **loadlistoftowns**() qui lit le fichier postesSynop.json de la base de données et retourne un dictionnaire (clé = nom d'une ville, valeur = code SYNOP).
    - Utiliser le module json de Python pour lire le fichier

Tester les fonctions au fur et à mesure.

Etape 1

- Module meteo1t.py
  - Définir une fonction **menu**() affichant sur une console les options suivantes :

    1 - Chercher une ville -> ce choix permet ensuite de saisir une chaine de caractères à rechercher

    2 – Sélectionner une ville -> ce choix permet ensuite de saisir le nom d'une ville

    3 - Afficher les données de la ville à une date donnée -> ce choix permet ensuite de saisir une date au format DD/MM/YYYY

    4 - Quitter

    La fonction demande un choix à l'utilisateur (1 à 4) et retourne ce choix accompagné de l'information associée à ce choix (ie elle retourne un tuple ; exemples : 1, 'par' ou 2, 'paris' ou 3, '31/07/2019' ou 4, None).
  - Définir une fonction **main**() qui, selon le choix de l'utilisateur :
    - Affiche la liste des villes et leur code contenant la sous-chaine saisie, sans être sensible à la casse (choix 1)
    - Charge les données du fichier SYNOP correspondant à la ville indiquée (choix 2)
    - Affiche les mesures de la ville saisie lors du choix 2 pour la date saisie (choix 3)
    - Quitte le programme (choix 4)
    - Boucle (ie affiche de nouveau le menu) tant que le choix n'est pas 4

    Note : bien gérer les erreurs qui pourraient survenir (exemple : faire le choix 3 avant de faire le choix 2, saisir une date invalide, saisir une ville inexistante dans la base…)

Etape 2

- Objectif de l'étape 2 : Créer une application permettant de rechercher et afficher les épisodes de canicule dans une ville sur tout l'historique disponible.
- On définit un épisode de canicule de la manière arbitraire suivante :
  - 1 jour caniculaire (dog day) si Tmin >= 20°C et Tmax >= 30°C ce jour là.
  - Episode de canicule (heat wave) si au moins 3 jours caniculaires consécutifs.

- Module meteo2.py :
  - Recopier meteo1.py en meteo2.py
  - Définir une fonction **finddogdays**(data: list) qui recherche les journées caniculaires d'une ville.
    - L'argument data est la liste des mesures pour une ville
    - La fonction retourne la liste des mesures appartenant à une journée caniculaire, regroupées par journée (elle retourne donc une liste de listes)
    - Attention, l'algorithme doit prendre en compte le fait que des mesures sont manquantes dans la base : il n'y a pas systématiquement 8 mesures par jour (un relevé peut manquer, auquel cas la ligne est absente du fichier).
  - Tester cette fonction. Pour tester commencer par choisir une ville ayant présenté des épisodes de canicule dans la période considérée (ex : Lyon). Vérifier ensuite qu'une ville sans canicule (ex : Brest) ne fait pas planter l'algorithme.

Etape 2

- Module meteo2.py (suite) :
  - Définir une fonction **findheatwaves**(data: list) qui recherche les épisodes de canicule d'une ville.
    - L'argument data est la liste des mesures pour une ville
    - La fonction commence par calculer la liste des jours caniculaires (en utilisant finddogdays), puis fait des regroupements de journées caniculaires consécutives. On peut utiliser timedelta() du module datetime, qui permet de mesurer l'écart entre deux dates ; ex : if date2 – date1 > timedelta(hours=…)…
    - La fonction retourne la liste des mesures appartenant à un épisode de canicule, regroupées par journée, et regroupées par épisodes de canicule (elle retourne donc une liste de listes de listes 😰)
  - Tester cette fonction

Structure du résultat retourné par findheatwaves() :

episode : resultat[0]									episode : resultat[1]								
journée : resultat[0][0]			journée : resultat[0][1]			journée : resultat[0][2]			journée : resultat[1][0]			journée : resultat[1][1]			journée : resultat[1][2]		
mesure resultat[0][0][0]	mesure resultat[0][0][1]	mesure resultat[0][0][2]	mesure resultat[0][1][0]	mesure resultat[0][1][1]	mesure resultat[0][1][1]	mesure resultat[0][2][0]	mesure resultat[0][2][1]	mesure resultat[0][2][2]	mesure resultat[1][0][0]	mesure resultat[1][0][1]	mesure resultat[1][0][2]	mesure resultat[1][1][0]	mesure resultat[1][1][1]	mesure resultat[1][1][2]	mesure resultat[1][2][0]	mesure resultat[1][2][1]	mesure resultat[1][2][2]

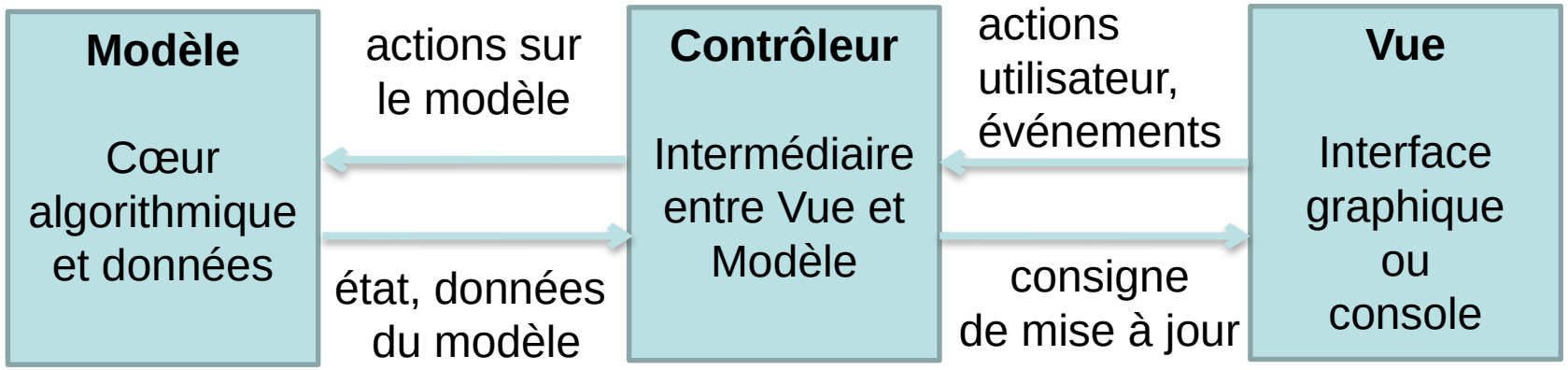
Etape 2

- Module meteo2t.py
  - Recopier meteo1t.py en meteo2t.py
  - Modifier le programme pour avoir le menu suivant :

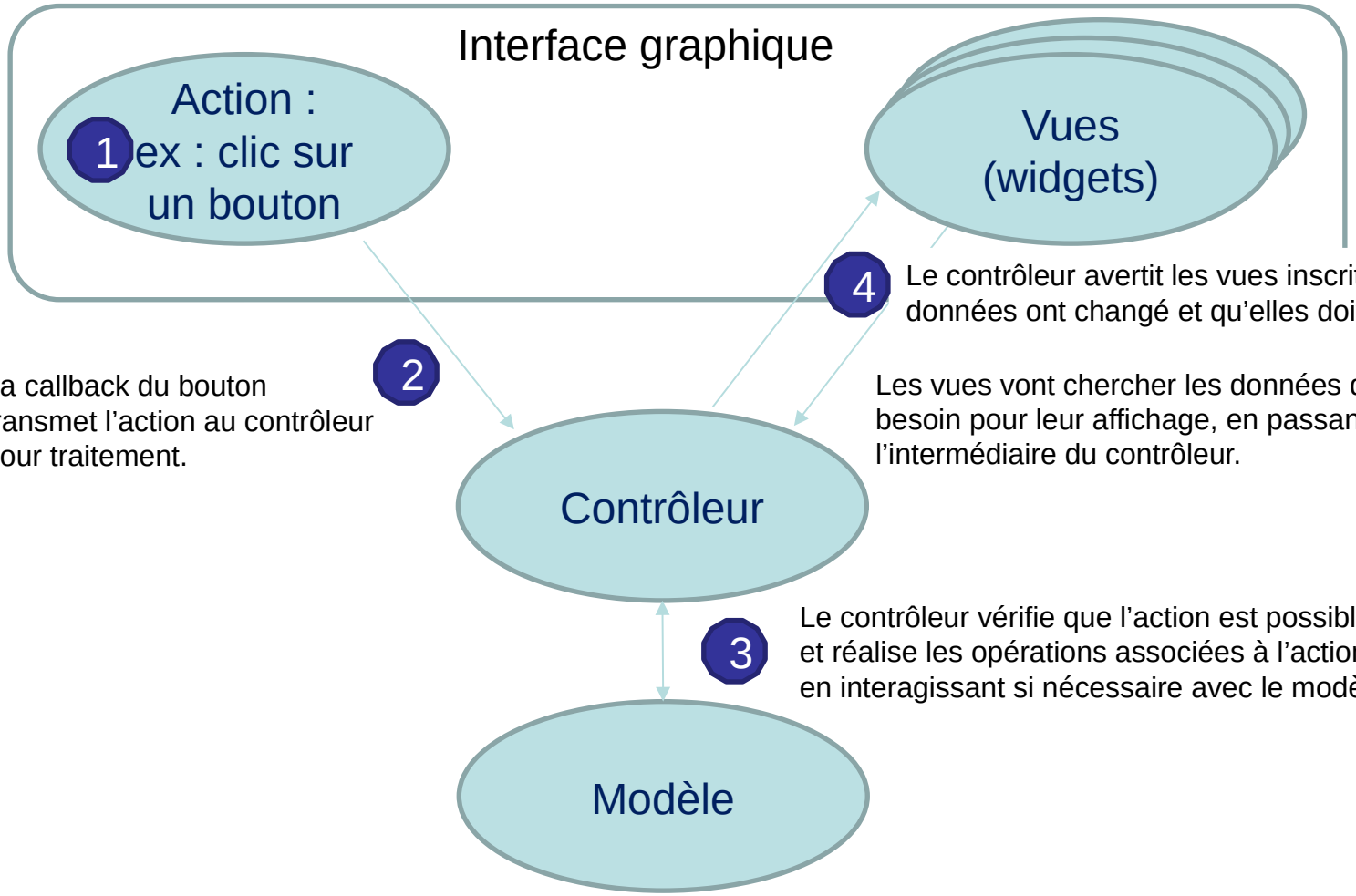
    1 - Chercher une ville
    2 - Sélectionner une ville
    3 - Afficher les données de la ville à une date donnée
    4 - Lister les canicules de la ville
    5 - Quitter
  - Implémenter l'option 4. Obtenir ce type d'affichage :

    5 épisode(s) de canicule trouvé(s) pour ORLY
    canicule 1 : 3 jours du 1998-08-09 au 1998-08-11
    canicule 2 : 10 jours du 2003-08-04 au 2003-08-13
    canicule 3 : 3 jours du 2017-07-06 au 2017-07-08
    canicule 4 : 3 jours du 2018-07-25 au 2018-07-27
    canicule 5 : 3 jours du 2019-06-25 au 2019-06-27

Architecture MVC : principe



Architecture MVC : exemple



Architecture MVC : code

```
class ControllerBase:
    def __init__(self):
        self.clients = []

    def inscrire(self, client):
        self.clients.append(client)

    def avertir(self):
        for client in self.clients:
            client.rafraichir()
```

À dériver en une classe Controller spécifique au projet

```
class ElementVue(QWidget):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.controller.inscrire(self)

    def rafraichir(self):
        …
```

Accès au contrôleur via self.controller

Etape 3

- Objectif de l'étape 3 : créer une interface graphique pour l'application (en utilisant Qt), en commençant par la visualisation de la liste des villes et la sélection d'une ville.
- Télécharger le fichier TEMPLATE_VUE.py et le renommer meteo3g.py
- Télécharger le fichier TEMPLATE_CONTROLEUR.py et le renommer meteo3c.py
- Ce template définit l'architecture de l'interface graphique, en subdivisant une fenêtre en 3 parties (à l'aide de layouts) : widget des paramètres (ParamsWidget), widget des graphiques (ChartsWidget) et widget de log (LogWidget).

- Module meteo3c.py
  - Constructeur de Controler : Créer un attribut self.towns (dictionnaire des villes) et l'initialiser en utilisant les fonctionnalités de meteo2.py (fonction loadlistoftowns)

- Module meteo3g.py
  - Dans le widget ParamsWidget, ajouter une liste déroulante contenant la liste des villes
    - Créer un layout vertical
    - Créer un objet QComboBox et le placer dans le layout
    - Définir une méthode initlistoftowns() qui ajoute les villes au QComboBox. La liste des villes est fournie par le contrôleur. Trier la liste dans l'ordre alphabétique.

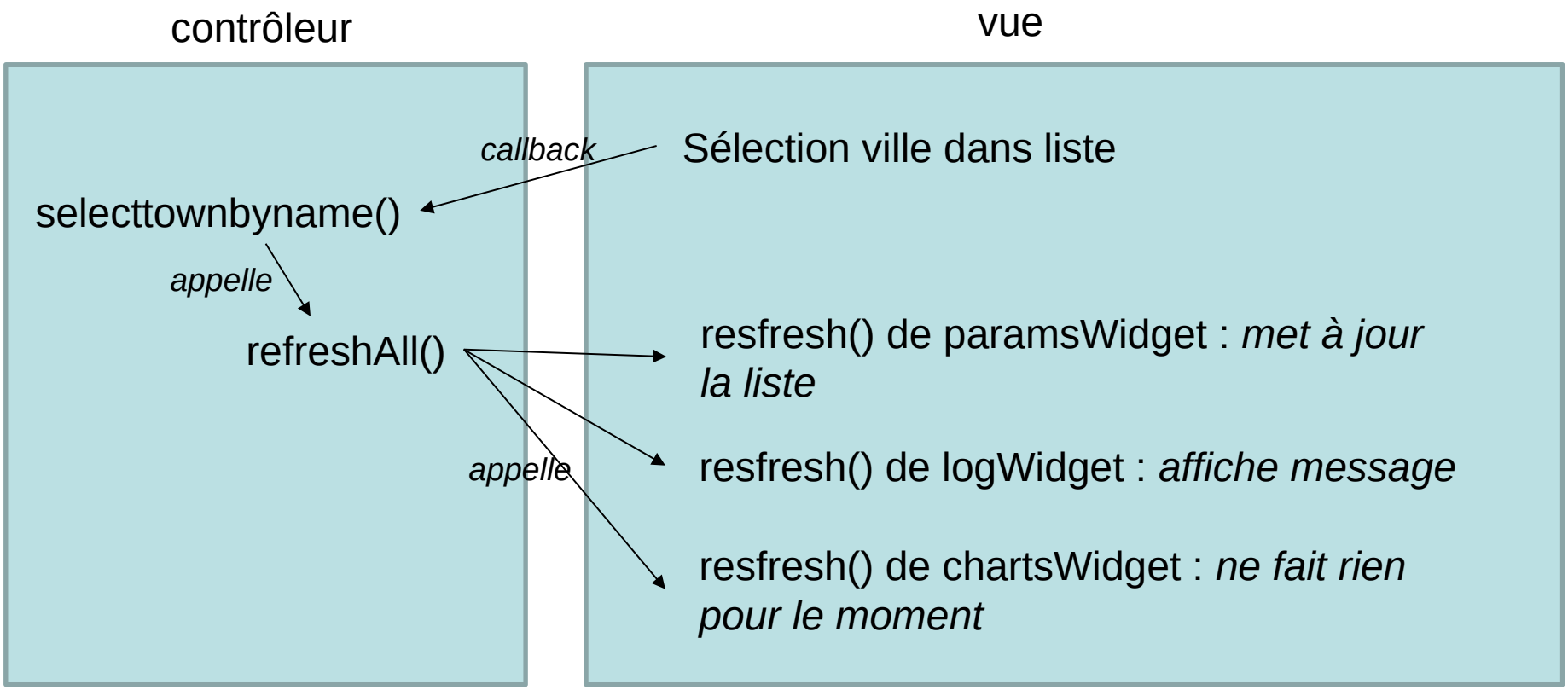
Etape 3

- Module meteo3c.py, classe Controler
  - Constructeur : définir les attributs suivants initialisés à None :
    - self.town: str
    - self.data: list
  - Définir une méthode selecttownbyname(name: str)
    - Cette fonction reçoit le nom d'une ville, modifie la ville courante (self.town), charge les données de la ville (self.data) en utilisant les fonctionnalités de meteo2.py, et prévient la vue (interface graphique) que la ville courante a changé (refreshAll).

- Module meteo3g.py
  - Installer la callback associée à la liste de villes. Cette callback doit activer la fonction selecttownbyname du controleur. Voir schéma page suivante.

- Module meteo3c.py, classe Controler
  - Méthode init() : appeler la fonction selecttownbyname('une ville au choix') pour définir la ville initialement sélectionnée au lancement de l'application

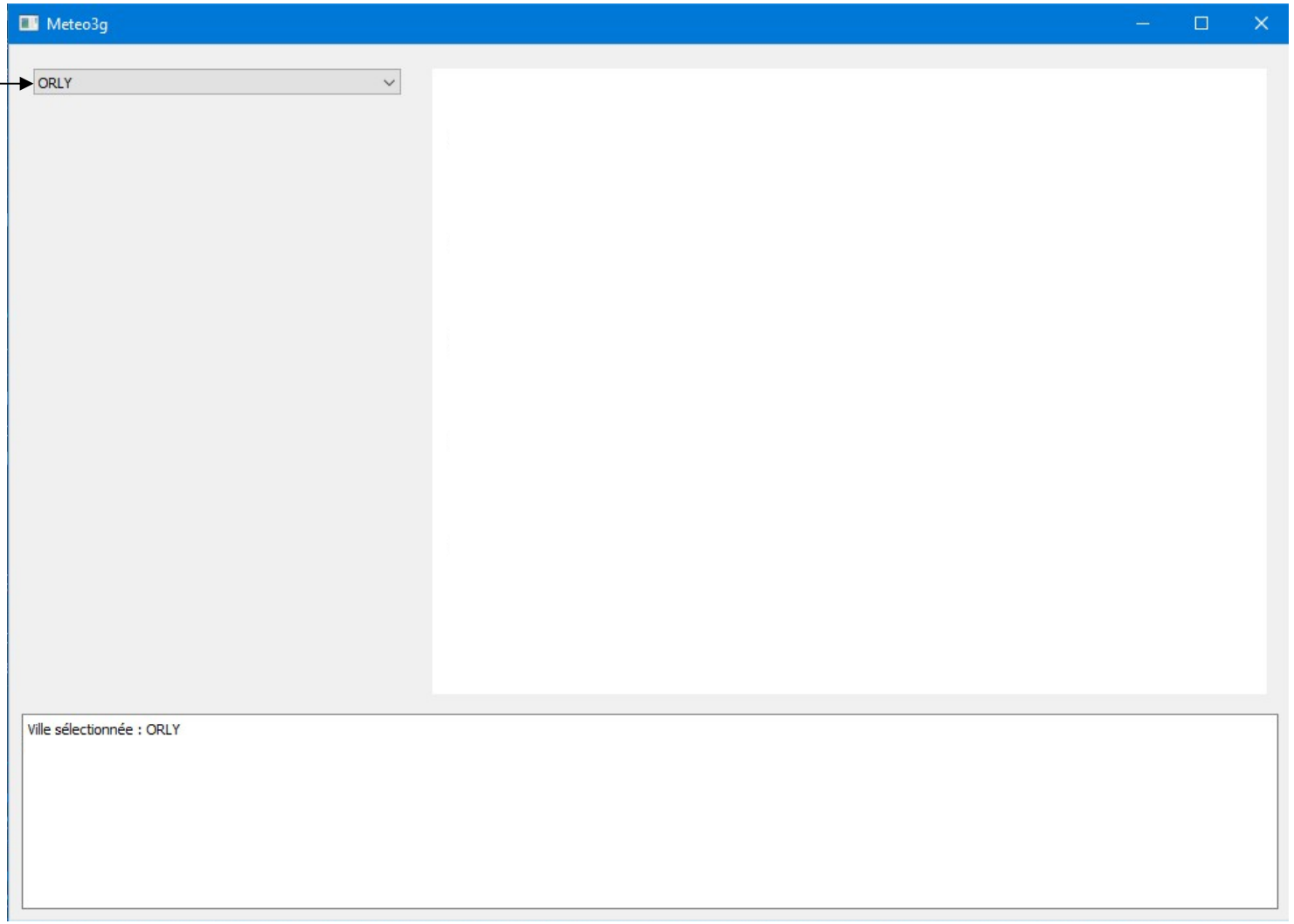
Etape 3



Flot de traitement d'un événement dans la vue (clic, sélection…)
événement dans vue -> traitement par contrôleur -> mise à jour vue

Etape 3

QComboBox



Etape 4

- Objectif de l'étape 4 : ajouter à la vue l'affichage des relevés météo pour une ville et une date sélectionnée.
- Recopier meteo3g.py en meteo4g.py et meteo3c.py en meteo4c.py
- Ajouter au widget ParamsWidgets un widget QCalendar qui permet de sélectionner une date.
  - Définir les dates minimum et maximum (du 01/01/1996 au 31/12/2019).
- Ajouter au contrôleur un attribut date: datetime et une méthode selectdate(date: datetime)
  - Cette méthode stocke la date reçue dans l'attribut self.date et prévient la vue que la date courante a changé (refreshAll).
- Connecter (callback) un click sur une date du calendrier à la fonction selectdate() du contrôleur.
  - Un message : « Date sélectionnée : xx/xx/xxxx » doit s'afficher dans LogWidget quand on change de date avec le widget QCalendar.
- Implémenter les méthodes refresh() et drawmeteo(self, data: list, town: str, date: datetime) de la classe ChartsWidget.
  - refresh() récupère les données data, town, date du contrôleur et appelle la méthode drawmeteo()
  - drawmeteo() trace les courbes de température et de précipitations pour la ville et le jour sélectionné. Un exemple de résultat attendu est présenté page suivante.

Etape 4