06/09/2019

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 « géant » 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 2018, à intervalle régulier de 3 heures (~67000 relevés par ville).

- La base de données initiale est structurée de la manière suivante :
  - 1 répertoire par année
  - 1 fichier CSV par mois comprenant les données des 62 villes

  *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 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 un tuple nommé Measure comprenant les 3 champs 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 (via un bloc try).
  - Définir une fonction date2path(date: datetime) qui retourne le chemin complet (chaine de caractères) du fichier SYNOP correspondant à la date fournie. La fonction vérifie l'existence du fichier, et retourne None au lieu du chemin s'il n'existe pas (utiliser os.path.isfile()).
  - Définir une fonction readsynopfile(filename: str) qui lit dans le fichier csv filename les données de toutes les villes.
    - Implémenter dans cette fonction la lecture du fichier csv fourni en argument (filename) en utilisant les fonctions open(), readline(), split().
    - La fonction retourne un dictionnaire : une clé est un identifiant de ville (code à 5 chiffres) et la valeur associée est une liste contenant les mesures de la ville (liste de tuples 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.
  - Pour les plus rapides : réécrire la fonction readsynopfile() en utilisant la classe DictReader du module csv.
  - Tester les fonctions (au fur et à mesure).

Etape 1

- Module meteo1.py
  - Définir un tuple nommé Town comprenant les 4 champs suivants : code, name, latitude, longitude.
  - Définir une fonction loadlistoftowns() qui lit le fichier postesSynop.json de la base de données et retourne une liste de tuples Town.
    - Utiliser le module json pour lire le fichier
  - Définir une fonction gettownbyname(towns: list, name: str) qui retourne une ville (un tuple Town) à partir du nom de la ville (name). Elle utilise la liste des villes disponibles (towns). Retourner None si le nom est invalide.
  - Tester ces fonctions.
- Module meteo1t.py
  - Définir une fonction menu() affichant sur une console les options suivantes :
    - 1 - Rechercher une ville -> ce choix permet ensuite de saisir une chaine de caractères à rechercher
    - 2 - Choisir une date -> ce choix permet ensuite de saisir une date au format DD/MM/YYYY
    - 3 - Afficher les données d'une ville -> ce choix permet ensuite de saisir le nom d'une ville
    - 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).
  - Définir une fonction main() qui, selon le choix de l'utilisateur :
    - Affiche la liste des villes et leur code contenant la sous-chaine saisie (choix 1)
    - Charge les données du fichier SYNOP correspondant à la date indiquée (choix 2)
    - Affiche les mesures de la ville saisie pour la date saisie lors du choix 2 (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 loadtowndata(code: str) qui charge tous les relevés météo d'une ville donnée.
    - Elle retourne une liste de mesures (liste de tuples Measure). Toutes les données de la base de données pour la ville sont prises en compte (toutes les années disponibles, tous les mois) et concaténées dans une liste unique de mesures.
    - La liste des années disponibles (1996 à 2018 inclus) et des mois (1 à 12) est définie comme une constante globale du module.
    - On constate à ce stade que le temps nécessaire pour charger toutes les données d'une ville est très long, car la structure de la base de données impose de parcourir toutes les données (~1 Go) alors que les données d'une seule ville sont légères (~17 Mo).
  - Ecrire un script convertdb.py qui restructure la base de données par ville plutôt que par année et par mois. La nouvelle base de données contient 62 fichiers csv (1 par ville) nommés avec le code de la ville (code_ville.csv ; exemple : 07005.csv), dans un répertoire database_by_town. La structure des fichiers csv est conservée (mêmes champs). Chaque fichier contient la totalité des données de la ville (toutes les années), soit environ 67000 lignes.
  - Définir une fonction loadtowndata_fast(code: str) qui charge tous les relevés météo d'une ville donnée, en utilisant la nouvelle base de données database_by_town.

Etape 2

- Module 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 (liste fournie par la fonction loadtowndata_fast)
    - 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.
  - 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 (liste fournie par la fonction loadtowndata_fast)
    - 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 ->cette option demande ensuite de saisir le nom d'une ville

    3 - Afficher les données de la ville à une date donnée -> demande ensuite à saisir une date

    4 - Lister les canicules de la ville

    5 - Quitter
  - Modifier le programme pour implémenter les options 2 et 3
    - Utiliser désormais loadtowndata_fast() pour charger les données de la ville sélectionnée.
  - Implémenter l'option 4. Obtenir ce type d'affichage :

    4 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

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 : Créer un attribut self.towns (liste 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.
  - Dans le widget ParamsWidget, afficher une carte interactive permettant d'afficher et de sélectionner les villes
    - Télécharger et recopier le package MapWidget dans le dossier de votre projet (conserver le répertoire MapWidget)
    - Importer (import) la classe MapWidget de ce package dans meteo3g.py
      - Méthodes utiles de la classe MapWidget : addMarker(), highlightMarker(), setZoom(), setCenter(), signal clicked()
    - Créer un objet MapWidget et l'ajouter au layout de ParamsWidget
    - Modifier initlistoftowns() pour ajouter des markers sur la carte pour chaque ville
    - Définir une fonction resetmapzoom() qui définit le zoom et le point central de la carte
    - Créer et positionner un bouton permettant de réinitialiser le zoom de la carte (une callback le connecte à la fonction resetmapzoom)

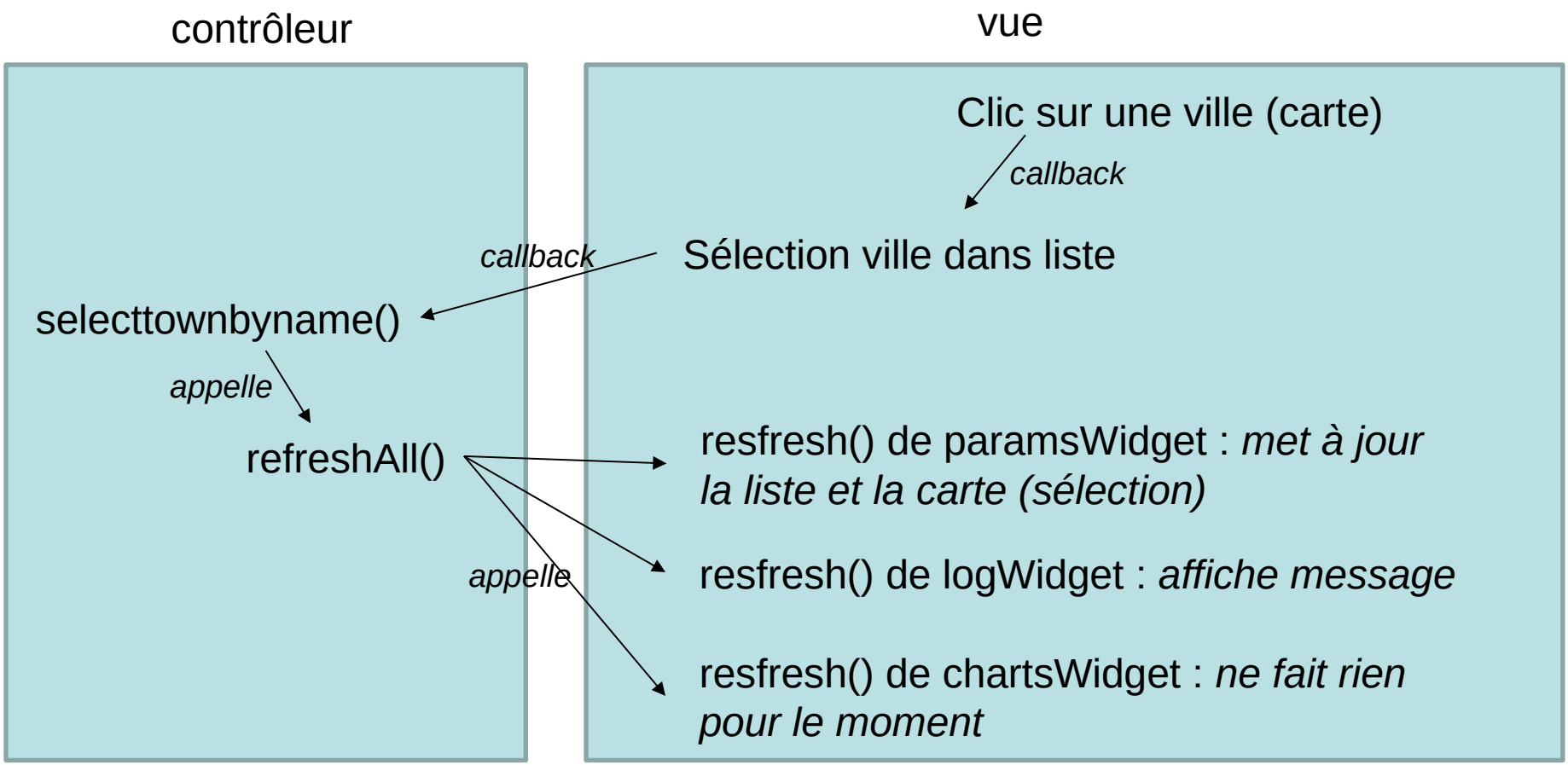
Etape 3

- Module meteo3c.py, classe Controler
  - Constructeur : définir les attributs suivants initialisés à None :
    - self.town: Town
    - 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 (gettownbyname, loadtowndata_fast), et prévient la vue (interface graphique) que la ville courante a changé (refreshAll).

- Module meteo3g.py
  - Installer les callbacks nécessaires à la synchronisation de la liste de ville, de la carte (ville en surbrillance synchrone avec la ville sélectionnée de la liste). Ces callbacks doivent 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



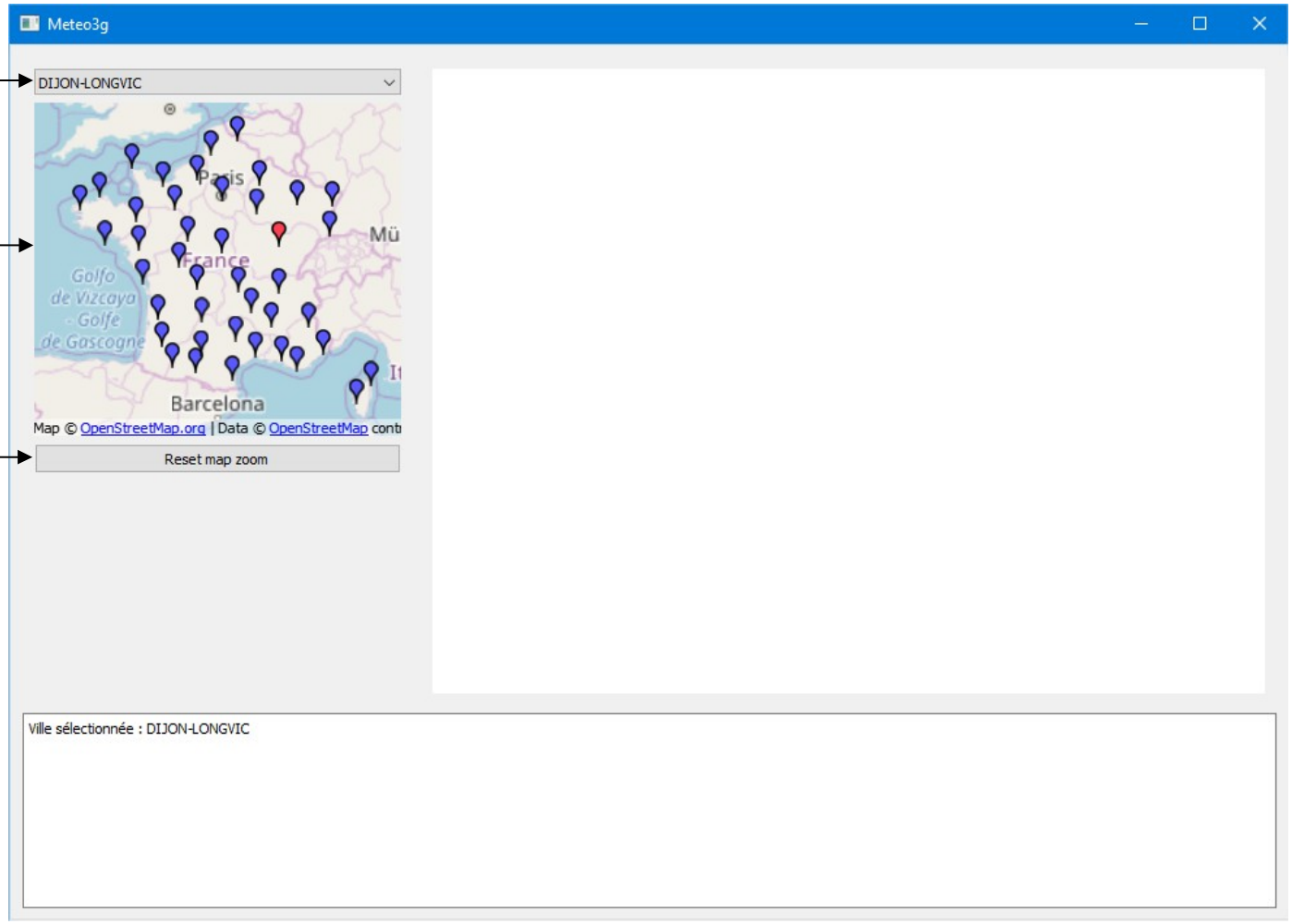
Flot de traitement d'un événement dans la vue (clic, sélection…)
événement dans vue -> traitement par contrôleur -> mise à jour vue

Etape 3

QComboBox

MapWidget

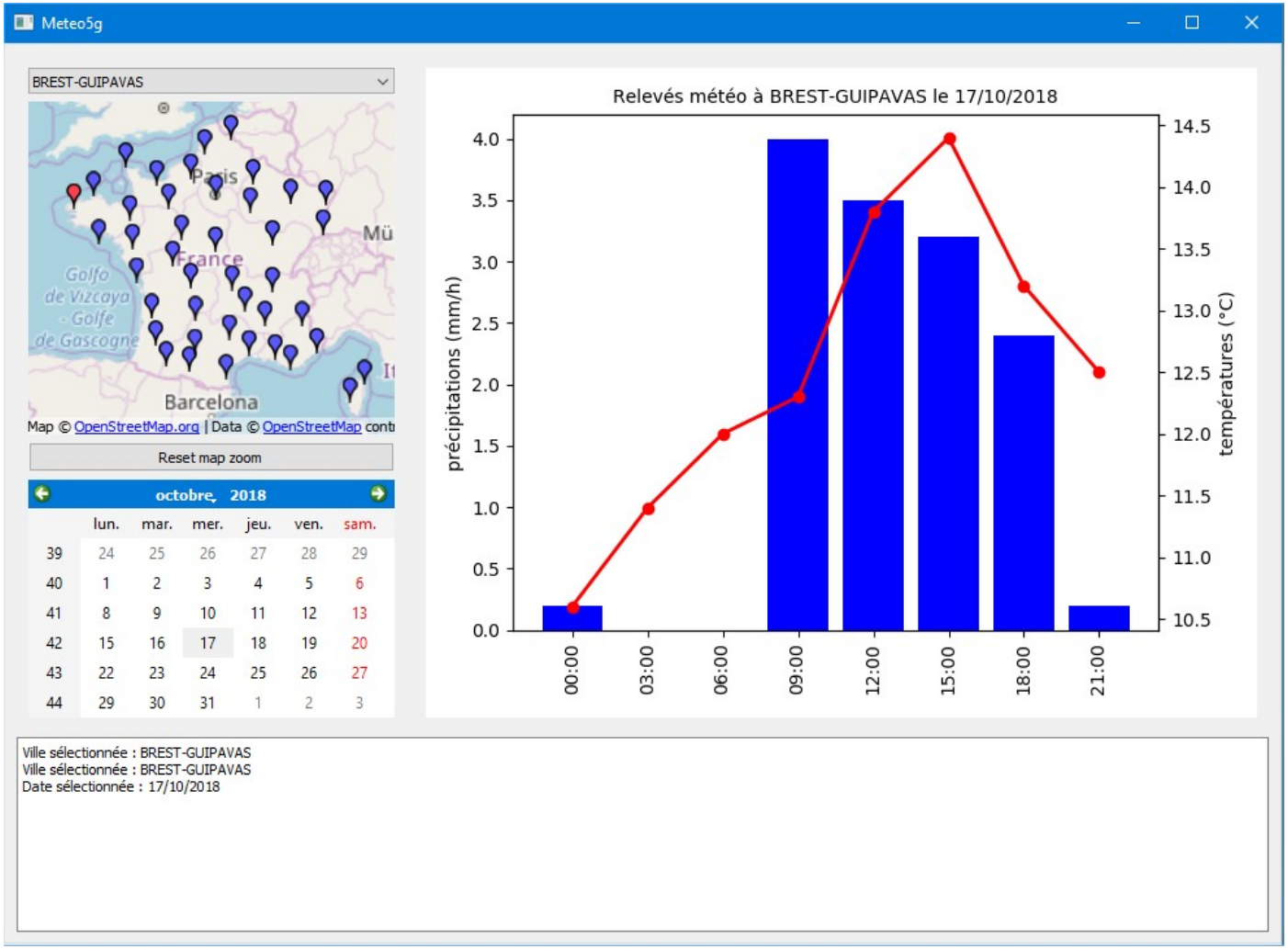
QPushButton



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 31/12/2018).
- 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: Town, 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



Etape 5

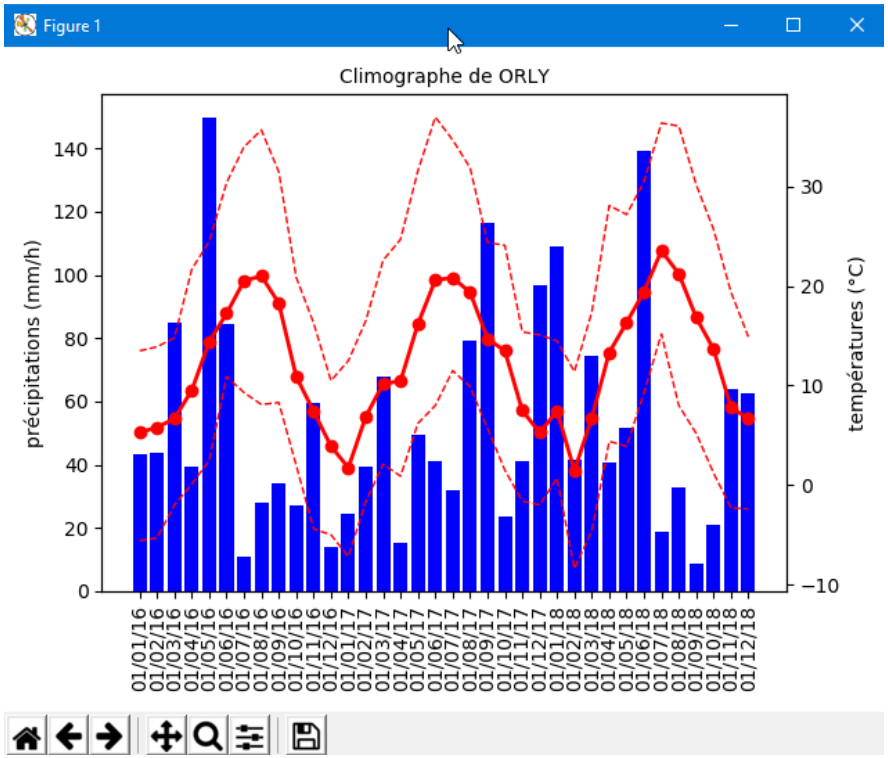
- Objectifs de l'étape 5 : Calculer des statistiques météo et les afficher dans une figure Matplotlib (hors Qt pour le moment)

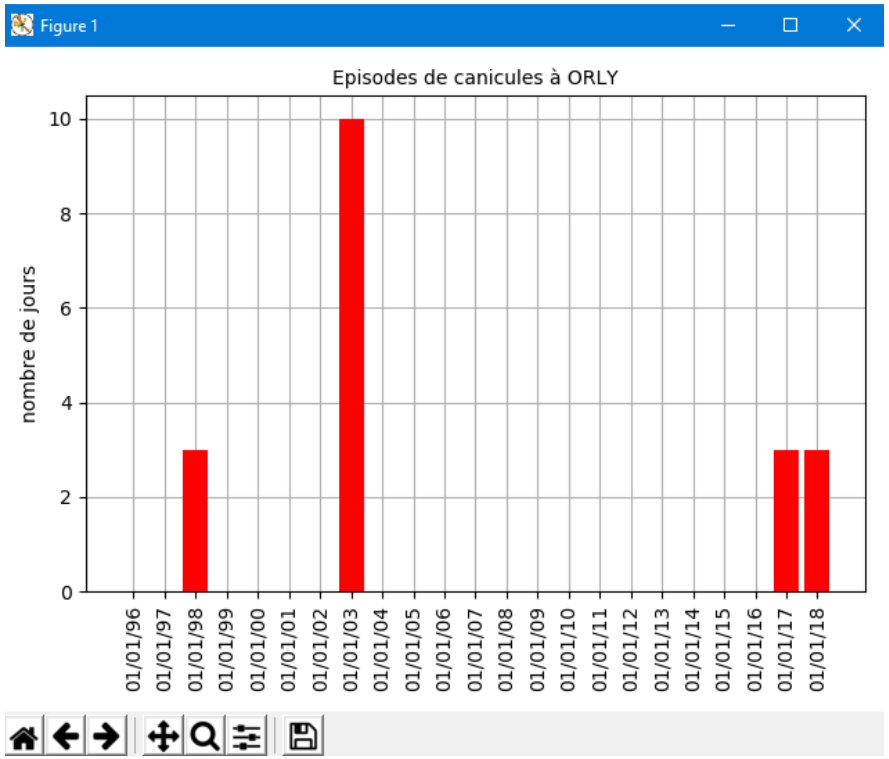
- Module meteo5.py
  - Copier meteo2.py en meteo5.py
  - Définir un nouveau type de tuple nommé : Stat(avgtemp: float, mintemp: float, maxtemp: float, cumrain: float)
  - Définir une fonction getstatonperiod(data: list, start: datetime, stop: datetime)
    - Cette fonction calcule des moyennes de températures et des cumuls de précipitations sur la période spécifiée par start (inclus) et stop (exclu).
    - L'argument data contient les données d'une ville (il provient de loadtowndata_fast)
    - La fonction retourne un tuple nommé de type Stat
    - Exclure des statistiques les températures = 0K qui sont en fait des données manquantes (cf. étape 1).
  - Tester cette fonction

Etape 5

- Module meteo5g.py *(NB : partir de 0, dans cette étape on n'utilise pas Qt)*
  - Définir une fonction daterange(start: datetime, stop: datetime, step: relativedelta)
    - Cette fonction retourne une liste de dates (datetime) comprises entre start (inclus) et stop (exclus) avec un pas de step (c'est donc l'équivalent de range() pour des dates)
    - step est de type relativedelta, défini dans le package dateutil
      - from dateutil.relativedelta import relativedelta
      - Exemple : step = relativedelta(months=3)
  - Définir une fonction drawclimograph(data: list, town: Town, start: datetime, stop: datetime, step: relativedelta)
    - Cette fonction construit et affiche une figure avec Matplotlib
    - Elle trace le graphique des températures et des précipitations sur la période spécifiée, avec le pas spécifiée. Utiliser la fonction getstatonperiod() pour obtenir la moyenne des températures et le cumul des précipitations sur l'intervalle 'step'.
  - Définir une fonction drawheatwaves(data: list, town: Town, start: datetime, stop: datetime, step: relativedelta)
    - Cette fonction construit et affiche l'histogramme des jours de canicule sur la période spécifiée avec Matplotlib
    - Utiliser la fonction findheatwaves()
  - Tester ces fonctions (au fur et à mesure). Résultat attendu page suivante.

Etape 5





Etape 6

- Objectif de l'étape 6
  - Intégrer les graphiques de l'étape 5 dans l'Ihm Qt
  - Ajouter une liste de choix du type de graphique à afficher
  - Ajouter la possibilité de définir la période de temps (start, stop, step)

- Modules meteo6g.py et meteo6c.py
  - Repartir des fichier meteo4g.py et meteo4c.py
  - Utiliser meteo5.py comme modèle
  - Intégrer les 2 graphiques de l'étape 5 à la vue (résultat attendu page suivante) :
    - Ajouter une liste déroulante à ParamsWidget permettant de choisir le type de graphique souhaité (=> 3 choix possibles)
    - Ajouter 3 widgets à ParamsWidgets permettant de définir une période de temps (année de début, année de fin, pas en mois)
    - Ajouter 2 méthodes de tracé (1 pour chaque type de graph supplémentaire) à ChartsWidget
    - Gérer l'ensemble à l'aide du contrôleur et des callbacks.

Etape 6

QComboBox
pour choix du graph

QSpinBox et
QComboBox
pour choix de la période