code: appli

```
'''from kivy.app import App
from kivy.uix.button import Button
from kivy.graphics import Color, RoundedRectangle
from kivy.uix.screenmanager import Screen, ScreenManager,
NoTransition

from Fil_actualites_screen import NewsFeedScreen
from Pronote_screen import PronoteScreen
from Home_screen import HomeScreen
from Rappels_screen import RappelsScreen
from Calendar_screen import CalendarScreen



class CustomButton(Button):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.background_normal=''
        self.background_color = (0, 0, 0, 0)
        self.color = (1, 1, 1, 1)

        with self.canvas.before:
            Color(0, 0.5, 1, 1)
            self.rounded_rect = RoundedRectangle(size=self.size,
pos=self.pos, radius=[50])

        self.bind(size=self.update_rounded_rect,
pos=self.update_rounded_rect)


    def update_rounded_rect(self, *args):
        self.rounded_rect.size = self.size
        self.rounded_rect.pos = self.pos

class CalendarScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = FloatLayout()

        with layout.canvas.before:
            Color(1, 1, 1, 1)  # Blanc
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création du bouton retour en haut
        back_button = CustomButton(
```

```
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': 1, 'right': 0.142}
        )

        # Changer l'arrondi uniquement pour ce bouton
        back_button.rounded_rect.radius = [20]  # Par exemple, 20 au
lieu de 50

        back_button.bind(on_press=self.goto_home)

        label = Label(
            text="Calendrier",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y':0.77}

        layout.add_widget(back_button)
        layout.add_widget(label)
        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def goto_home(self, instance):
        self.manager.current = 'home'

class NewsFeedScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = FloatLayout()

        with layout.canvas.before:
            Color(1, 1, 1, 1)
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création du bouton retour
```

```
        back_button = CustomButton(
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': 1, 'right': 0.142}
        )

        # Changer l'arrondi uniquement pour ce bouton
        back_button.rounded_rect.radius = [20]  # Par exemple, 20 au
lieu de 50

        back_button.bind(on_press=self.goto_home)

        # Titre
        label = Label(
            text="Fil d'Actualité",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y': 0.77}

        layout.add_widget(back_button)
        layout.add_widget(label)
        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def goto_home(self, instance):
        self.manager.current = 'home'

class HomeScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = BoxLayout(orientation='vertical', padding=50,
spacing=20)

        with layout.canvas.before:
            Color(1, 1, 1, 1)  # Fond blanc
            self.rect = Rectangle(size=self.size, pos=self.pos)
        self.bind(size=self.update_rect, pos=self.update_rect)
```

```
        # Création des boutons avec leurs événements
        calendar_button = CustomButton(
            text="Calendrier",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )
        news_feed_button = CustomButton(
            text="Fil d'Actualité",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )
        pronote_button = CustomButton(
            text="Pronote",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )
        rappels_button = CustomButton(
            text="Rappels",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )

        # Lier les boutons aux fonctions de navigation
        calendar_button.bind(on_press=self.goto_calendar)
        news_feed_button.bind(on_press=self.goto_news_feed)
        pronote_button.bind(on_press=self.goto_pronote)
        rappels_button.bind(on_press=self.goto_rappels)

        button_grid = GridLayout(cols=2, spacing=20, padding=50)

        # Ajout des widgets au layout
        layout.add_widget(Widget(size_hint_y=1))  # Espace vide en
haut
        layout.add_widget(button_grid)
        layout.add_widget(Widget(size_hint_y=0.5))  # Espace en bas
```

```
        # Ajout des boutons au grid
        button_grid.add_widget(calendar_button)
        button_grid.add_widget(pronote_button)
        button_grid.add_widget(news_feed_button)
        button_grid.add_widget(rappels_button)

        self.add_widget(layout)

    def goto_calendar(self, instance):
        self.manager.current = 'calendar'

    def goto_news_feed(self, instance):
        self.manager.current = 'news_feed'

    def goto_pronote(self, instance):
        self.manager.current = 'pronote'

    def goto_rappels(self, instance):
        self.manager.current = 'rappels'

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

class RappelsScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = FloatLayout()

        with layout.canvas.before:
            Color(1, 1, 1, 1)  # Blanc
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création du bouton retour en haut
        back_button = CustomButton(
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': 1, 'right': 0.142}
        )

        # Changer l'arrondi uniquement pour ce bouton
        back_button.rounded_rect.radius = [20]  # Par exemple, 20 au
lieu de 50
```

```
        back_button.bind(on_press=self.goto_home)

        label = Label(
            text="Rappels",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y': 0.77}

        layout.add_widget(back_button)
        layout.add_widget(label)
        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def goto_home(self, instance):
        self.manager.current = 'home'


class PronoteScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = FloatLayout()

        with layout.canvas.before:
            Color(1, 1, 1, 1)  # Blanc
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création du bouton retour en haut
        back_button = CustomButton(
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': 1, 'right': 0.142}
        )

        # Changer l'arrondi uniquement pour ce bouton
```

```
        back_button.rounded_rect.radius = [20]  # Par exemple, 20 au
lieu de 50

        back_button.bind(on_press=self.goto_home)

        label = Label(
            text="Pronote",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y': 0.77}

        layout.add_widget(back_button)
        layout.add_widget(label)
        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def goto_home(self, instance):
        self.manager.current = 'home'


class MainApp(App):
    def build(self):
        # Création du gestionnaire de screens
        sm = ScreenManager(transition=NoTransition())
        sm.add_widget(HomeScreen(name='home'))
        sm.add_widget(CalendarScreen(name='calendar'))
        sm.add_widget(NewsFeedScreen(name='news_feed'))
        sm.add_widget(RapelsScreen(name='rappels'))
        sm.add_widget(PronoteScreen(name='pronote'))
        return sm


if __name__ == '__main__':
    MainApp().run()'''
```

code calendar screen

```
import json
from kivy.app import App
```

```
from plyer import orientation

from parametres import *
from kivy.uix.label import Label
from kivy.uix.widget import Widget
from kivy.core.window import Window
from kivy.uix.screenmanager import Screen
from kivy.uix.gridlayout import GridLayout
from kivy.graphics import Color, Rectangle
from kivy.uix.boxlayout import BoxLayout
from Custom_button_screen import CustomButton

# --- Vos données JSON ---
json_data = '''
{
   "schedule": [
       {
           "start_time": "8h30",
           "end_time": "10h25",
           "subject": "ED.PHYSIQUE & SPORT.",
           "teacher": "GALLAND L.",
           "room": "",
           "status": "",
           "color": "#195A46"
       },
       {
           "start_time": "10h40",
           "end_time": "11h35",
           "subject": "SCIENCES VIE & TERRE",
           "teacher": "CORDIER Z.",
           "room": "salle 203",
           "status": "Prof. absent",
           "color": "#212853"
       },
       {
           "start_time": "11h35",
           "end_time": "12h30",
           "subject": "ANGLAIS LV1",
           "teacher": "BERTRAND C.",
           "room": "salle 213",
           "status": "",
           "color": "#FFED00"
       },
       {
           "start_time": "12h30",
           "end_time": "13h55",
           "subject": "Pas de cours",
           "teacher": "",
```

```
            "room": "",
            "status": "",
            "color": ""
        },
        {
            "start_time": "13h55",
            "end_time": "14h50",
            "subject": "ESPAGNOL LV2",
            "teacher": "SORIANO I.",
            "room": "salle 304",
            "status": "",
            "color": "#ED679B"
        },
        {
            "start_time": "15h10",
            "end_time": "17h00",
            "subject": "MEDIA",
            "teacher": "SORIANO I.",
            "room": "[305_MEDIA]",
            "status": "salle 211",
            "color": "#A2C62B"
        }
    ]
}
'''
data = json.loads(json_data)
schedule_data = data['schedule']


# --- Création du tableau du planning ---
class ScheduleTable(GridLayout):
    def __init__(self, schedule, **kwargs):
        super().__init__(**kwargs)
        self.cols = 5  # Une colonne pour chaque champ sauf
"Couleur"
        self.spacing = 5
        self.padding = 5

        # Ligne d'entête
        headers = ["Horaire", "Matière", "Professeur", "Salle",
"Statut"]
        for header in headers:
            header_label = Label(text=header, bold=True,
size_hint_y=None, height=40)
            self.add_widget(header_label)

        # Ajout des horaires pour chaque cours
        for item in schedule:
```

```
            # Combiner les colonnes Début et Fin en une seule
colonne avec "de ... à ..."
            time_range = f"de {item['start_time']} à
{item['end_time']}"
            self.add_widget(Label(text=time_range,
size_hint_y=None, height=40))
            self.add_widget(Label(text=item['subject'],
size_hint_y=None, height=40))
            self.add_widget(Label(text=item['teacher'],
size_hint_y=None, height=40))

            # Vérifier si la salle est "[305_MEDIA]", et l'enlever
si c'est le cas
            room = item['room']
            if "[305_MEDIA]" in room:
                room = ""  # On ne l'affiche pas dans la colonne
"Salle"

            # Affichage de la salle (si différente de
"[305_MEDIA]")
            self.add_widget(Label(text=room, size_hint_y=None,
height=40))

            # Affichage du statut, si présent (dans ce cas, salle
211 est dans le statut)
            status = item['status']
            self.add_widget(Label(text=status, size_hint_y=None,
height=40))


# --- Écran Calendrier ---
class CalendarScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.background_color = (1, 1, 1, 1)  # Fond blanc

        layout = BoxLayout(orientation='vertical')

        # Titre
        label = Label(
            text="Calendrier",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)  # Texte en noir
```

```
        )
        label.pos_hint = {'x': 0.15, 'y': 0.85}

        # Ajouter une ligne de séparation (couvrant toute la
largeur de l'écran)
        separator = Widget(size_hint=(None, None),
size=(Window.width, 2.5), pos_hint={'x': 0, 'y': 0.85})
        with separator.canvas:
            Color(0, 0, 0, 1)  # Noir
            self.sep_rect = Rectangle(pos=separator.pos,
size=separator.size)
        separator.bind(pos=self.update_sep, size=self.update_sep)

        # Tableau des horaires
        schedule_table = ScheduleTable(schedule_data,
                                       size_hint=(1, None),  #
Taille dynamique
                                       height=Window.height * 0.6,
# Limiter la hauteur à 60%
                                       pos_hint={'x': 0.05, 'y':
0.1})  # Placer le tableau avec un peu d'espace

        # Assurer que le texte du tableau est noir
        schedule_table.color = (0, 0, 0, 1)  # Texte du tableau en
noir

        # Création du bouton retour
        back_button = CustomButton(
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': 0, 'right': 1}  # Le placer en bas à
droite
        )

        # Ajouter une bordure arrondie au bouton retour
        with back_button.canvas.before:
            Color(0.7, 0.7, 0.7, 1)  # Couleur de fond du bouton
(gris clair)
            self.rect = Rectangle(size=back_button.size,
pos=back_button.pos, radius=[20])
        back_button.bind(on_press=self.goto_home)

        # Ajouter les éléments au layout
        layout.add_widget(label)  # Ajouter d'abord le titre
        layout.add_widget(separator)  # Ajouter la ligne de
séparation
```

```
        layout.add_widget(schedule_table)  # Ajouter le tableau
        layout.add_widget(back_button)  # Ajouter le bouton retour
en bas

        self.add_widget(layout)

    def update_sep(self, instance, value):
        # Cette méthode est appelée lorsque la position ou la
taille du séparateur change
        self.sep_rect.pos = instance.pos
        self.sep_rect.size = instance.size

    def goto_home(self, instance):
        self.manager.current = 'home'
# Lancement de l'application
class MyApp(App):
    def build(self):
        return CalendarScreen()  # Pas de 'name' ici
```

code connexion_pronote:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time
import logging
import sys
import json
import re
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.service import Service
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from bs4 import BeautifulSoup

# ------------------------------
# Configuration du logging
# ------------------------------
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    stream=sys.stdout
)
```

```
logger = logging.getLogger("ScrapPronote")

# ------------------------------
# Fonction d'attente d'un élément
# ------------------------------
def wait_for_element(driver, by, selector, wait_time=20):
    """Attend la présence d'un élément sur la page."""
    try:
        return WebDriverWait(driver, wait_time).until(
            EC.presence_of_element_located((by, selector))
        )
    except TimeoutException:
        logger.warning("Timeout lors de l'attente de l'élément :
%s", selector)
        return None

# ------------------------------
# Classe de connexion à Pronote
# ------------------------------
class ScrapPronote:
    def __init__(self):
        self.USERNAME = "ethan.trunsard"
        self.PASSWORD = "ARfox324#384268"
        self.URL_LOGIN = (

"https://www.moncollege-ent.essonne.fr/auth/login?callback=%2Fcas%
2Flogin%3Fservice%3D"

"https%253A%252F%252F0910860R.index-education.net%252Fpronote%252F
eleve.html#/"
        )
        self.WAIT_TIME = 20

    def login(self, driver):
        logger.info("Accès à la page de connexion Pronote...")
        driver.get(self.URL_LOGIN)
        time.sleep(3)  # Pause initiale

        email_field = wait_for_element(driver, By.NAME, "email",
self.WAIT_TIME)
        password_field = wait_for_element(driver, By.NAME,
"password", self.WAIT_TIME)

        if not email_field or not password_field:
            raise Exception("Les champs de connexion ne sont pas
accessibles.")

        email_field.send_keys(self.USERNAME)
```

```
        password_field.send_keys(self.PASSWORD)
        password_field.send_keys(Keys.RETURN)
        logger.info("Formulaire de connexion soumis.")

        # Attendre que la page d'accueil s'affiche (exemple avec
une liste de cours)
        if not wait_for_element(driver, By.CSS_SELECTOR,
"ul.liste-cours", self.WAIT_TIME):
            raise Exception("La page d'accueil Pronote ne semble
pas chargée correctement.")
        logger.info("Connexion réussie, page d'accueil chargée.")

# ------------------------------
# Fonctions d'extraction des données
# ------------------------------

def parse_schedule(soup):
    """Extrait le planning depuis la section 'liste-cours'."""
    schedule = []
    ul = soup.find("ul", class_="liste-cours")
    if ul:
        for li in ul.find_all("li", class_="flex-contain",
recursive=False):
            # Récupérer le texte caché qui contient l'heure et la
matière
            sr_text = li.find("span",
class_="sr-only").get_text(strip=True) if li.find("span",
class_="sr-only") else ""
            # Exemple attendu : "de 8h30 à 10h25 ED.PHYSIQUE &
SPORT."
            match =
re.search(r"de\s+(\d{1,2}h\d{2})\s+à\s+(\d{1,2}h\d{2})\s+(.*)",
sr_text)
            if match:
                start_time = match.group(1)
                end_time = match.group(2)
                subject_text = match.group(3)
            else:
                start_time = end_time = subject_text = ""
            # Récupération des informations dans la liste des cours
            container = li.find("ul", class_="container-cours")
            items = [item.get_text(strip=True) for item in
container.find_all("li")] if container else []
            # On suppose que le premier élément est la matière
(redondant avec sr_text), le deuxième le professeur,
            # le troisième éventuellement la salle, et le quatrième
un statut (ex: "Prof. absent")
            teacher = items[1] if len(items) > 1 else ""
```

```
            room = items[2] if len(items) > 2 else ""
            status = items[3] if len(items) > 3 else ""
            # Récupération de la couleur dans le style du
trait-matiere
            trait = li.find("div", class_="trait-matiere")
            color = ""
            if trait and trait.has_attr("style"):
                style = trait["style"]
                m = re.search(r"background-color\s*:\s*([^;]+)",
style)
                if m:
                    color = m.group(1).strip()
            schedule.append({
                "start_time": start_time,
                "end_time": end_time,
                "subject": subject_text,
                "teacher": teacher,
                "room": room,
                "status": status,
                "color": color
            })
    return schedule

def parse_homework(soup):
    """Extrait les devoirs depuis la section correspondante."""
    homework_list = []
    # On cherche le conteneur identifié par id_96 (par exemple)
    container = soup.find("div", id="id_96")
    if container:
        # Les devoirs sont dans une liste imbriquée
        li_items = container.find_all("li")
        for li in li_items:
            # Extraire le sujet depuis <span class="titre-matiere">
            subject_el = li.find("span", class_="titre-matiere")
            # Extraire le statut depuis <div class="text
ie-ellipsis">
            status_el = li.find("div", class_="text",
recursive=True)
            # Extraire la description depuis <div
class="description">
            desc_el = li.find("div", class_="description")
            # Pour la date, on pourrait essayer de récupérer le
titre du groupe parent (<h3>) si disponible
            date_el = li.find_parent("li",
attrs={"aria-labelledby":True})
            date_text = ""
            if date_el and date_el.find("h3"):
                date_text = date_el.find("h3").get_text(strip=True)
```

```
            homework_list.append({
                "subject": subject_el.get_text(strip=True) if
subject_el else "",
                "status": status_el.get_text(strip=True) if
status_el else "",
                "description": desc_el.get_text(strip=True) if
desc_el else "",
                "date": date_text
            })
    return homework_list

def parse_notes(soup):
    """Extrait les notes depuis la section 'liste-clickable'."""
    notes = []
    ul = soup.find("ul", class_="liste-clickable")
    if ul:
        for li in ul.find_all("li", recursive=False):
            a = li.find("a", class_="wrapper-link")
            if a:
                subject_el = a.find("h3")
                date_el = a.find("div",
class_="infos-conteneur").find("span", class_="date") if
a.find("div", class_="infos-conteneur") else None
                eval_container = a.find("div",
class_="evaluations-conteneur")
                evaluations = []
                if eval_container:
                    for span in eval_container.find_all("span",
recursive=True):
                        title = span.get("title", "").strip()
                        style = span.get("style", "")
                        color = ""
                        m =
re.search(r"background-color\s*:\s*([^;]+)", style)
                        if m:
                            color = m.group(1).strip()
                        evaluations.append({
                            "title": title,
                            "color": color,
                            "text": span.get_text(strip=True)
                        })
                notes.append({
                    "subject": subject_el.get_text(strip=True) if
subject_el else "",
                    "date": date_el.get_text(strip=True) if date_el
else "",
                    "evaluations": evaluations
                })
```

```
    return notes

def time_to_minutes(t):
    """Convertit un temps au format '8h30' en minutes."""
    try:
        parts = t.split("h")
        if len(parts) == 2:
            hours = int(parts[0])
            minutes = int(parts[1])
            return hours * 60 + minutes
    except Exception:
        pass
    return 0

# ------------------------------
# Main
# ------------------------------
def main():
    # Configuration du driver Firefox
    gecko_path =
r"C:/Users/Ethan/Downloads/geckodriver-v0.36.0-win64/geckodriver.e
xe"
    service = Service(gecko_path)
    driver = webdriver.Firefox(service=service)
    scrap = ScrapPronote()

    try:
        # Connexion à Pronote via Selenium
        scrap.login(driver)
        time.sleep(5)  # Attendre que le JavaScript ait terminé de
charger la page

        # Récupérer le code HTML dynamique (avec page_source)
        html = driver.page_source

        # Parser le HTML avec BeautifulSoup
        soup = BeautifulSoup(html, "html.parser")
        # Optionnel : supprimer les balises <script> si vous ne
souhaitez pas les conserver
        for script in soup.find_all("script"):
            script.decompose()
        # Sauvegarder le HTML nettoyé dans un fichier (UTF-8)
        with open("code_html_pronote_sans_script.html", "w",
encoding="utf-8") as f:
            f.write(str(soup))
        logger.info("Le code HTML dynamique a été récupéré et
enregistré dans 'code_html_pronote_sans_script.html'.")
```

```
        # Extraction des données
        schedule = parse_schedule(soup)
        homework = parse_homework(soup)
        notes = parse_notes(soup)

        # Trier le planning par heure de début
        schedule.sort(key=lambda x:
time_to_minutes(x.get("start_time", "")))

        data = {
            "schedule": schedule,
            "homework": homework,
            "notes": notes
        }

        # Sauvegarder les données extraites dans un fichier JSON
(UTF-8)
        with open("extracted_data.json", "w", encoding="utf-8") as
f:
            json.dump(data, f, ensure_ascii=False, indent=4)
        logger.info("Les données ont été extraites et sauvegardées
dans 'extracted_data.json'.")
        print(json.dumps(data, ensure_ascii=False, indent=4))

    except Exception as e:
        logger.error("Une erreur s'est produite : %s", e)
    finally:
        driver.quit()
        logger.info("Driver fermé.")

if __name__ == "__main__":
    main()
```

code Custom_buttun screen 👍

```
from kivy.uix.button import Button
from kivy.graphics import Color, RoundedRectangle

class CustomButton(Button):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.background_normal=''
        self.background_color = (0, 0, 0, 0)
        self.color = (1, 1, 1, 1)

        with self.canvas.before:
            Color(0, 0.5, 1, 1)
            self.rounded_rect = RoundedRectangle(size=self.size,
pos=self.pos, radius=[50])
```

```
        self.bind(size=self.update_rounded_rect,
pos=self.update_rounded_rect)


    def update_rounded_rect(self, *args):
        self.rounded_rect.size = self.size
        self.rounded_rect.pos = self.pos
```

code json extracted dat {

```
    "schedule": [
        {
            "start_time": "8h30",
            "end_time": "10h25",
            "subject": "ED.PHYSIQUE & SPORT.",
            "teacher": "GALLAND L.",
            "room": "",
            "status": "",
            "color": "#195A46"
        },
        {
            "start_time": "10h40",
            "end_time": "11h35",
            "subject": "SCIENCES VIE & TERRE",
            "teacher": "CORDIER Z.",
            "room": "salle 203",
            "status": "Prof. absent",
            "color": "#212853"
        },
        {
            "start_time": "11h35",
            "end_time": "12h30",
            "subject": "ANGLAIS LV1",
            "teacher": "BERTRAND C.",
            "room": "salle 213",
            "status": "",
            "color": "#FFED00"
        },
        {
            "start_time": "12h30",
            "end_time": "13h55",
            "subject": "Pas de cours",
            "teacher": "",
            "room": "",
            "status": "",
            "color": ""
        },
```

```
        {
            "start_time": "13h55",
            "end_time": "14h50",
            "subject": "ESPAGNOL LV2",
            "teacher": "SORIANO I.",
            "room": "salle 304",
            "status": "",
            "color": "#ED679B"
        },
        {
            "start_time": "15h10",
            "end_time": "17h00",
            "subject": "MEDIA",
            "teacher": "SORIANO I.",
            "room": "[305_MEDIA]",
            "status": "salle 211",
            "color": "#A2C62B"
        }
    ],
    "homework": [
        {
            "subject": "ARTS PLASTIQUES",
            "status": "Fait",
            "description": "Si vous avez des magazines dans
lesquels vous pourriez découper, n'hésitez pas à les apporter
jeudi !",
            "date": ""
        },
        {
            "subject": "ARTS PLASTIQUES",
            "status": "Fait",
            "description": "Si vous avez des magazines dans
lesquels vous pourriez découper, n'hésitez pas à les apporter
jeudi !",
            "date": "Pouraujourd'hui"
        },
        {
            "subject": "HISTOIRE-GEOGRAPHIE-EMC",
            "status": "Fait",
            "description": "Contrôle H4.",
            "date": "Pouraujourd'hui"
        },
        {
            "subject": "ANGLAIS LV1",
            "status": "Non Fait",
            "description": "Copier le cours déposé dans l’ent",
            "date": ""
        },
```

```
        {
            "subject": "ANGLAIS LV1",
            "status": "Non Fait",
            "description": "Copier le cours déposé dans l’ent",
            "date": "Pourdemain"
        },
        {
            "subject": "ESPAGNOL LV2",
            "status": "Fait",
            "description": "Apportez votre ordinateur chargé pour
l'évaluation",
            "date": "Pourdemain"
        },
        {
            "subject": "ESPAGNOL LV2",
            "status": "Non Fait",
            "description": "Prepara la evaluación del final de la
secuencia",
            "date": "Pourdemain"
        },
        {
            "subject": "ESPAGNOL LV2",
            "status": "Non Fait",
            "description": "Prepara la evaluación del final de la
secuencia.Puedes escuchar de nuevo mi comentario oral sobre la
evaluacion precedente",
            "date": "Pourdemain"
        },
        {
            "subject": "VIE DE CLASSE",
            "status": "Fait",
            "description": "Rendre son rapport de stage relié et
dactylographié (format papier). Vous pouvez bien entendu le rendre
avant. Aucun retard ne sera accepté.",
            "date": "Pourdemain"
        },
        {
            "subject": "FRANCAIS",
            "status": "Non Fait",
            "description": "Faire sur feuille correctement
présentée :Rédigez la question de synthèse.Faire l’exercice sur
les personnifications.",
            "date": ""
        },
        {
            "subject": "FRANCAIS",
            "status": "Non Fait",
```

```
            "description": "Faire sur feuille correctement
présentée :Rédigez la question de synthèse.Faire l’exercice sur
les personnifications.",
            "date": "Pourlundi 10 mars"
        },
        {
            "subject": "MATHEMATIQUES",
            "status": "Non Fait",
            "description": "79p41",
            "date": "Pourlundi 10 mars"
        },
        {
            "subject": "SCIENCES VIE & TERRE",
            "status": "Non Fait",
            "description": "Évaluation fiche mémo : vous serez
interrogés sur 1 à 3 questions parmi celles des fiches de
mémorisation n°1, n°2 et n°3",
            "date": ""
        },
        {
            "subject": "SCIENCES VIE & TERRE",
            "status": "Non Fait",
            "description": "Évaluation fiche mémo : vous serez
interrogés sur 1 à 3 questions parmi celles des fiches de
mémorisation n°1, n°2 et n°3",
            "date": "Pourmardi 11 mars"
        }
    ],
    "notes": [
        {
            "subject": "FRANCAIS",
            "date": "le 4 mars",
            "evaluations": [
                {
                    "title": "Très bonne maîtrise - Lire des œuvres
littéraires et fréquenter des œuvres d'art.",
                    "color": "",
                    "text": "+"
                },
                {
                    "title": "",
                    "color": "#008000",
                    "text": "+"
                },
                {
                    "title": "Presque maîtrisé - Élaborer une
interprétation de textes littéraires.",
                    "color": "",
```

```
                    "text": ""
                },
                {
                    "title": "",
                    "color": "#ADDE1F",
                    "text": ""
                }
            ]
        },
        {
            "subject": "LES METHODES ET OUTILS POUR APPRENDRE",
            "date": "le 4 mars",
            "evaluations": [
                {
                    "title": "Maîtrise satisfaisante - Faire et
rendre un travail pour la date demandée",
                    "color": "",
                    "text": ""
                },
                {
                    "title": "",
                    "color": "#45B851",
                    "text": ""
                }
            ]
        },
        {
            "subject": "ESPAGNOL LV2",
            "date": "le 25 févr.",
            "evaluations": [
                {
                    "title": "Très bonne maîtrise - Comprendre les
idées principales de documents courts.",
                    "color": "",
                    "text": "+"
                },
                {
                    "title": "",
                    "color": "#008000",
                    "text": "+"
                },
                {
                    "title": "Maîtrise fragile - Repérer des
informations ciblées.",
                    "color": "",
                    "text": ""
                },
                {
```

```
                    "title": "",
                    "color": "#FFDA01",
                    "text": ""
                }
            ]
        },
        {
            "subject": "ANGLAIS LV1",
            "date": "le 21 févr.",
            "evaluations": [
                {
                    "title": "Maîtrise satisfaisante - Écrire un
court récit, des poèmes simples...",
                    "color": "",
                    "text": ""
                },
                {
                    "title": "",
                    "color": "#45B851",
                    "text": ""
                },
                {
                    "title": "Presque maîtrisé - EOC - Lire à haute
voix un texte très bref.",
                    "color": "",
                    "text": ""
                },
                {
                    "title": "",
                    "color": "#ADDE1F",
                    "text": ""
                },
                {
                    "title": "Maîtrise satisfaisante - La
prononciation est compréhensible, l’intonation et l’accentuation
sont généralement correctes.",
                    "color": "",
                    "text": ""
                },
                {
                    "title": "",
                    "color": "#45B851",
                    "text": ""
                }
            ]
        },
        {
            "subject": "MEDIA",
```

```
            "date": "le 14 févr.",
            "evaluations": [
                {
                    "title": "Très bonne maîtrise - S'impliquer,
s'engager.",
                    "color": "",
                    "text": "+"
                },
                {
                    "title": "",
                    "color": "#008000",
                    "text": "+"
                }
            ]
        },
        {
            "subject": "EDUCATION MUSICALE",
            "date": "le 3 févr.",
            "evaluations": [
                {
                    "title": "Maîtrise satisfaisante - Échanger,
partager, argumenter et débattre",
                    "color": "",
                    "text": ""
                },
                {
                    "title": "",
                    "color": "#45B851",
                    "text": ""
                }
            ]
        },
        {
            "subject": "ESPAGNOL LV2",
            "date": "le 3 févr.",
            "evaluations": [
                {
                    "title": "Très bonne maîtrise - Comprendre les
idées principales de documents courts.",
                    "color": "",
                    "text": "+"
                },
                {
                    "title": "",
                    "color": "#008000",
                    "text": "+"
                }
            ]
```

```
        }
    ]
}
```

code fil actulaites `from kivy.uix.label import Label`

```
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Color, Rectangle
from kivy.uix.screenmanager import Screen
from kivy.uix.widget import Widget
from kivy.core.window import Window
from parametres import *
from Custom_button_screen import CustomButton

class NewsFeedScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = FloatLayout()

        with layout.canvas.before:
            Color(1, 1, 1, 1)
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création du bouton retour
        back_button = CustomButton(
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': horizontale_button_retour, 'right':
vertical_button_retour}
        )

        # Changer l'arrondi uniquement pour ce bouton
        back_button.rounded_rect.radius = [20]  # Par exemple, 20
au lieu de 50

        back_button.bind(on_press=self.goto_home)

        # Titre
        label = Label(
            text="Fil d'actualité",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
```

```
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y': 0.77}

        # Ajouter une ligne de séparation (couvrant toute la
largeur de l'écran)
        separator = Widget(size_hint=(None, None),
size=(Window.width, 2.5), pos_hint={'x': 0, 'y': 0.85})
        with separator.canvas:
            Color(0, 0, 0, 1)  # Noir
            self.sep_rect = Rectangle(pos=separator.pos,
size=separator.size)
        separator.bind(pos=self.update_sep, size=self.update_sep)

        # Ajouter les éléments au layout
        layout.add_widget(back_button)
        layout.add_widget(label)
        layout.add_widget(separator)

        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def update_sep(self, instance, value):
        # Met à jour la position et la taille du rectangle de
séparation
        self.sep_rect.pos = instance.pos
        self.sep_rect.size = instance.size

    def goto_home(self, instance):
        self.manager.current = 'home'
```

code home screen `from kivy.uix.widget import Widget`

```
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout

from kivy.graphics import Color, Rectangle
from kivy.uix.screenmanager import Screen
from Custom_button_screen import CustomButton

class HomeScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = BoxLayout(orientation='vertical', padding=50,
spacing=20)
```

```
        with layout.canvas.before:
            Color(1, 1, 1, 1)  # Fond blanc
            self.rect = Rectangle(size=self.size, pos=self.pos)
        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création des boutons avec leurs événements
        calendar_button = CustomButton(
            text="Calendrier",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )
        news_feed_button = CustomButton(
            text="Fil d'Actualité",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )
        pronote_button = CustomButton(
            text="Pronote",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )
        rappels_button = CustomButton(
            text="Rappels",
            font_size=24,
            size_hint=(None, None),
            size=(300, 100),
            background_normal='',
            background_color=(0, 0.5, 1, 1)
        )

        # Lier les boutons aux fonctions de navigation
        calendar_button.bind(on_press=self.goto_calendar)
        news_feed_button.bind(on_press=self.goto_news_feed)
        pronote_button.bind(on_press=self.goto_pronote)
        rappels_button.bind(on_press=self.goto_rappels)

        button_grid = GridLayout(cols=2, spacing=20, padding=50)
```

```
        # Ajout des widgets au layout
        layout.add_widget(Widget(size_hint_y=1))  # Espace vide en
haut
        layout.add_widget(button_grid)
        layout.add_widget(Widget(size_hint_y=0.5))  # Espace en bas

        # Ajout des boutons au grid
        button_grid.add_widget(calendar_button)
        button_grid.add_widget(pronote_button)
        button_grid.add_widget(news_feed_button)
        button_grid.add_widget(rappels_button)

        self.add_widget(layout)

    def goto_calendar(self, instance):
        self.manager.current = 'calendar'

    def goto_news_feed(self, instance):
        self.manager.current = 'news_feed'

    def goto_pronote(self, instance):
        self.manager.current = 'pronote'

    def goto_rappels(self, instance):
        self.manager.current = 'rappels'

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos
```

code main

```
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, NoTransition

from Home_screen import HomeScreen
from Rappels_screen import RappelsScreen
from Pronote_screen import PronoteScreen
from Calendar_screen import CalendarScreen
from Fil_actualites_screen import NewsFeedScreen

class MainApp(App):
    def build(self):
        # Création du gestionnaire de screens
        sm = ScreenManager(transition=NoTransition())
        sm.add_widget(HomeScreen(name='home'))
        sm.add_widget(CalendarScreen(name='calendar'))
```

```
        sm.add_widget(NewsFeedScreen(name='news_feed'))
        sm.add_widget(RappelsScreen(name='rappels'))
        sm.add_widget(PronoteScreen(name='pronote'))
        return sm


if __name__ == '__main__':
    MainApp().run()
```

code parametre

```
horizontale_button_retour = 0.99
vertical_button_retour = 0.15
horizontale_titre = 0.12
vertical_titre = 0.77
```

code pronote screen `from kivy.uix.label import Label`

```
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Color, Rectangle
from kivy.uix.screenmanager import Screen
from kivy.uix.widget import Widget
from kivy.core.window import Window
from parametres import *
from Custom_button_screen import CustomButton

class PronoteScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = FloatLayout()

        with layout.canvas.before:
            Color(1, 1, 1, 1)  # Blanc
            self.rect = Rectangle(size=self.size, pos=self.pos)

        self.bind(size=self.update_rect, pos=self.update_rect)

        # Création du bouton retour en haut
        back_button = CustomButton(
            text="Retour",
            font_size=30,
            size_hint=(None, None),
            size=(112.5, 75),
            pos_hint={'top': horizontale_button_retour, 'right':
vertical_button_retour}
        )

        # Changer l'arrondi uniquement pour ce bouton
```

```
        back_button.rounded_rect.radius = [20]  # Par exemple, 20
au lieu de 50

        back_button.bind(on_press=self.goto_home)

        # Titre
        label = Label(
            text="Pronote",
            font_size=80,
            size_hint=(None, None),
            size=(600, 200),
            text_size=(600, None),
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y': 0.77}

        # Ajouter une ligne de séparation (couvrant toute la
largeur de l'écran)
        separator = Widget(size_hint=(None, None),
size=(Window.width, 2.5), pos_hint={'x': 0, 'y': 0.85})
        with separator.canvas:
            Color(0, 0, 0, 1)  # Noir
            self.sep_rect = Rectangle(pos=separator.pos,
size=separator.size)
        separator.bind(pos=self.update_sep, size=self.update_sep)

        layout.add_widget(back_button)
        layout.add_widget(label)
        layout.add_widget(separator)

        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def update_sep(self, instance, value):
        # Met à jour la position et la taille du rectangle de
séparation
        self.sep_rect.pos = instance.pos
        self.sep_rect.size = instance.size

    def goto_home(self, instance):
        self.manager.current = 'home'
```

rappel screen `from kivy.uix.label import Label`

```
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Color, Rectangle
from kivy.uix.screenmanager import Screen
from kivy.uix.widget import Widget
from kivy.core.window import Window
from parametres import *
from Custom_button_screen import CustomButton

class RappelsScreen(Screen):
   def __init__(self, **kwargs):
       super().__init__(**kwargs)
       layout = FloatLayout()

       with layout.canvas.before:
           Color(1, 1, 1, 1)
           self.rect = Rectangle(size=self.size, pos=self.pos)

       self.bind(size=self.update_rect, pos=self.update_rect)

       # Création du bouton retour
       back_button = CustomButton(
           text="Retour",
           font_size=30,
           size_hint=(None, None),
           size=(112.5, 75),
           pos_hint={'top': horizontale_button_retour, 'right':
vertical_button_retour}
       )

       # Changer l'arrondi uniquement pour ce bouton
       back_button.rounded_rect.radius = [20]  # Par exemple, 20
au lieu de 50

       back_button.bind(on_press=self.goto_home)

       # Titre
       label = Label(
           text="Rappels",
           font_size=80,
           size_hint=(None, None),
           size=(600, 200),
           text_size=(600, None),
```

```
            halign='center',
            valign='middle',
            color=(0, 0, 0, 1)
        )
        label.pos_hint = {'x': 0.12, 'y': 0.77}

        # Ajouter une ligne de séparation (couvrant toute la
largeur de l'écran)
        separator = Widget(size_hint=(None, None),
size=(Window.width, 2.5), pos_hint={'x': 0, 'y': 0.85})
        with separator.canvas:
            Color(0, 0, 0, 1)  # Noir
            self.sep_rect = Rectangle(pos=separator.pos,
size=separator.size)
        separator.bind(pos=self.update_sep, size=self.update_sep)

        # Ajouter les éléments au layout
        layout.add_widget(back_button)
        layout.add_widget(label)
        layout.add_widget(separator)

        self.add_widget(layout)

    def update_rect(self, *args):
        self.rect.size = self.size
        self.rect.pos = self.pos

    def update_sep(self, instance, value):
        # Met à jour la position et la taille du rectangle de
séparation
        self.sep_rect.pos = instance.pos
        self.sep_rect.size = instance.size

    def goto_home(self, instance):
        self.manager.current = 'home'
```

code scrap `#!/usr/bin/env python`

```
# -*- coding: utf-8 -*-
"""
=============================================================================
=
ScrapPronote - Script complet et robuste pour récupérer les données
=============================================================================
=
```

```
Ce script utilise Selenium et Firefox pour se connecter à Pronote via
l'ENT et récupérer un maximum d'informations :

   - Emploi du temps : horaires, matières, professeurs, salles, et couleur.
   - Devoirs : date d'échéance et description.
   - Notes : matière, valeur de la note et couleur.

Le script inclut des mécanismes pour gérer le chargement dynamique du JS,
la navigation dans plusieurs onglets, des re-tentatives et un filtrage des cours vides.

ATTENTION :
   - Les identifiants sont en clair : modifie-les et sécurise-les après tests.
   - Vérifie et adapte les sélecteurs CSS selon la structure HTML réelle.

Historique :
- Version initiale : 2025-03-05
- Version améliorée par ChatGPT avec corrections et robustesse.

================================================================================
"""

# ================================================================
# IMPORTS ET CONFIGURATION GLOBALE
# ================================================================
import time
import json
import logging
import sys
import traceback

from typing import TextIO, Any
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.options import Options as FirefoxOptions
from selenium.webdriver.firefox.service import Service as FirefoxService
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```
from selenium.common.exceptions import (NoSuchElementException,
TimeoutException,
                                        WebDriverException)

# Configuration du logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    stream=sys.stdout
)
logger = logging.getLogger("ScrapPronote")


# ====================================================================
# FONCTIONS UTILITAIRES
# ====================================================================
def wait_for_element(driver, by, selector, wait_time=20):
    """Attend la présence d'un élément sur la page."""
    try:
        element = WebDriverWait(driver, wait_time).until(
            EC.presence_of_element_located((by, selector))
        )
        return element
    except TimeoutException:
        logger.warning("Timeout lors de l'attente de l'élément :
%s", selector)
        return None

def click_element(driver, by, selector, wait_time=20):
    """Tente de cliquer sur un élément dès qu'il est cliquable."""
    try:
        element = WebDriverWait(driver, wait_time).until(
            EC.element_to_be_clickable((by, selector))
        )
        element.click()
        return True
    except TimeoutException:
        logger.warning("Timeout lors du clic sur l'élément : %s",
selector)
        return False
    except Exception as e:
        logger.error("Erreur lors du clic sur l'élément %s: %s",
selector, e)
        return False

def retry_operation(func, max_retries=3, delay=5):
    """ Réessaye une opération en cas d'échec. """
    for attempt in range(max_retries):
```

```
        try:
            return func()
        except Exception as e:
            logger.warning("Tentative %d échouée: %s", attempt + 1,
e)
            time.sleep(delay)
    return None

def save_data_to_file(data, filename="pronote_data.json"):
    """Sauvegarde les données extraites dans un fichier JSON."""
    try:
        with open(filename, "w", encoding="utf-8") as f:
            json.dump(data, f, ensure_ascii=False, indent=4)
        logger.info("Données sauvegardées dans le fichier : %s",
filename)
    except Exception as e:
        logger.error("Erreur lors de la sauvegarde des données :
%s", e)


def parse_sr_only_text(text):
    """
    Parse le texte du span.sr-only pour extraire l'horaire et la
matière.
    Exemple attendu : "de 8h30 à 9h25 PHYSIQUE-CHIMIE"
    """
    result = {"horaire_debut": "", "horaire_fin": "", "matiere":
""}
    try:
        if text.lower().startswith("de ") and "à" in text.lower():
            parts = text.split("à")
            start_part = parts[0].replace("de", "").strip()  # ex:
"8h30"
            rest = parts[1].strip()  # ex: "9h25 PHYSIQUE-CHIMIE"
            subparts = rest.split(" ", 1)
            if len(subparts) == 2:
                result["horaire_debut"] = start_part
                result["horaire_fin"] = subparts[0].strip()
                result["matiere"] = subparts[1].strip()
    except Exception as e:
        logger.error("Erreur lors du parsing du texte sr-only :
%s", e)
    return result

def convert_color(rgb_string):
    """Convertit une couleur RGB (ex : "rgb(0, 153, 218)") en
hexadécimal."""
    try:
```

```
        rgb = rgb_string.replace("rgb(", "").replace(")",
"").split(",")
        rgb = [int(x.strip()) for x in rgb]
        return "#{:02X}{:02X}{:02X}".format(rgb[0], rgb[1], rgb[2])
    except Exception as e:
        logger.warning("Erreur lors de la conversion de couleur :
%s", e)
        return rgb_string

# ============================================================
# CLASSE PRINCIPALE : ScrapPronote
# ============================================================
class ScrapPronote:
    """
    Classe pour se connecter à Pronote et extraire un maximum
d'informations.
    Extrait :
      - Emploi du temps (horaires, matière, professeur, salle,
couleur)
      - Devoirs (date d'échéance, description)
      - Notes (matière, valeur, couleur)
    """

    def __init__(self):
        # Paramètres de connexion et URL
        self.USERNAME = "ethan.trunsard"
        self.PASSWORD = "ARfox324#384268"
        self.URL_LOGIN = (
            "https://www.moncollege-ent.essonne.fr/auth/login?"
            "callback=%2Fcas%2Flogin%3Fservice%3D"

"https%253A%252F%252F0910860R.index-education.net%252Fpronote%252F
eleve.html#/"
        )
        self.GECKODRIVER_PATH =
r"C:/Users/Ethan/Downloads/geckodriver-v0.36.0-win64/geckodriver.e
xe"
        self.WAIT_TIME = 20

        # Structure pour stocker les données extraites
        self.data = {
            "timetable": [],
            "homework": [],
            "notes": []
        }

    #
----------------------------------------------------------------
```

```
    # Méthode de connexion
    #
-------------------------------------------------------------------
    def login(self, driver):
        logger.info("Accès à la page de connexion Pronote...")
        driver.get(self.URL_LOGIN)
        time.sleep(3)  # Pause pour le chargement initial

        email_field = wait_for_element(driver, By.NAME, "email",
self.WAIT_TIME)
        password_field = wait_for_element(driver, By.NAME,
"password", self.WAIT_TIME)

        if not email_field or not password_field:
            raise Exception("Les champs de connexion ne sont pas
accessibles.")

        email_field.send_keys(self.USERNAME)
        password_field.send_keys(self.PASSWORD)
        password_field.send_keys(Keys.RETURN)
        logger.info("Formulaire de connexion soumis.")

        # Attendre que la page d'accueil soit chargée : on vérifie
la présence du conteneur des cours
        if not wait_for_element(driver, By.CSS_SELECTOR,
"ul.liste-cours", self.WAIT_TIME):
            raise Exception("La page d'accueil Pronote ne semble
pas chargée correctement.")
        logger.info("Connexion réussie, page d'accueil chargée.")

    #
-------------------------------------------------------------------
    # Extraction de l'emploi du temps
    #
-------------------------------------------------------------------
    def get_timetable_data(self, driver):
        logger.info("Début de l'extraction de l'emploi du
temps...")
        timetable_data = []
        try:
            # On cible le conteneur principal de la liste des cours
            courses = driver.find_elements(By.CSS_SELECTOR,
"ul.liste-cours li.flex-contain")
            logger.info("Nombre de cours trouvés: %s",
len(courses))
            for course in courses:
                course_info = {
                    "horaire_debut": "",
```

```
                    "horaire_fin": "",
                    "matiere": "",
                    "professeur": "",
                    "salle": "",
                    "couleur": ""
                }
                # Extraction des horaires depuis le
div.container-heures
                try:
                    container_heures =
course.find_element(By.CSS_SELECTOR, "div.container-heures")
                    heures =
container_heures.find_elements(By.TAG_NAME, "div")
                    if len(heures) >= 2:
                        course_info["horaire_debut"] =
heures[0].text.strip()
                        course_info["horaire_fin"] =
heures[1].text.strip()
                    else:
                        logger.warning("Nombre insuffisant de div
dans container-heures.")
                except NoSuchElementException:
                    logger.warning("Horaires non trouvés pour un
cours.")

                # Extraction de la couleur depuis le
div.trait-matiere
                try:
                    trait = course.find_element(By.CSS_SELECTOR,
"div.trait-matiere")
                    raw_color =
trait.value_of_css_property("background-color")
                    course_info["couleur"] =
convert_color(raw_color)
                except NoSuchElementException:
                    logger.warning("Trait couleur non trouvé pour
ce cours.")

                # Extraction des informations internes dans
ul.container-cours
                try:
                    inner_ul = course.find_element(By.CSS_SELECTOR,
"ul.container-cours")
                    inner_lis = inner_ul.find_elements(By.TAG_NAME,
"li")
                    if len(inner_lis) > 0:
                        course_info["matiere"] =
inner_lis[0].text.strip()
```

```
                    if len(inner_lis) > 1:
                        course_info["professeur"] =
inner_lis[1].text.strip()
                    if len(inner_lis) > 2:
                        course_info["salle"] =
inner_lis[2].text.strip()
                except NoSuchElementException:
                    logger.warning("Informations
matière/professeur/salle non trouvées pour un cours.")

                # Si les informations essentielles ne sont pas
extraites, tenter de parser le span.sr-only
                if not (course_info["horaire_debut"] or
course_info["matiere"]):
                    try:
                        sr_text =
course.find_element(By.CSS_SELECTOR, "span.sr-only").text.strip()
                        parsed = parse_sr_only_text(sr_text)
                        course_info["horaire_debut"] =
parsed.get("horaire_debut", "")
                        course_info["horaire_fin"] =
parsed.get("horaire_fin", "")
                        course_info["matiere"] =
parsed.get("matiere", "")
                    except NoSuchElementException:
                        logger.warning("Élément sr-only non trouvé
pour le cours.")

                # Filtrer les cours totalement vides (aucune info
essentielle)
                if not (course_info["matiere"] or
course_info["horaire_debut"] or course_info["professeur"]):
                    logger.info("Cours vide détecté, passage au
suivant.")
                    continue

                timetable_data.append(course_info)
                logger.info("Cours extrait: %s", course_info)
        except Exception as e:
            logger.error("Erreur lors de l'extraction de l'emploi
du temps: %s", e)
            traceback.print_exc()
        return timetable_data

    #
-----------------------------------------------------------------
    # Extraction des devoirs (cahier de texte)
```

```
    #
-----------------------------------------------------------------
    def get_homework_data(self, driver):
        logger.info("Début de l'extraction des devoirs...")
        homework_data = []
        try:
            # Navigation vers l'onglet "Cahier de texte" (à adapter
selon le HTML réel)
            if not click_element(driver, By.CSS_SELECTOR,
".menu-cahierdetexte", self.WAIT_TIME):
                logger.warning("Impossible de cliquer sur l'onglet
Devoirs/Cahier de texte.")
                return homework_data

            if not wait_for_element(driver, By.CSS_SELECTOR,
"#GInterface .conteneur-liste-CDT", self.WAIT_TIME):
                logger.warning("Liste des devoirs non trouvée.")
                return homework_data

            devoirs_elements =
driver.find_elements(By.CSS_SELECTOR, "#GInterface
.conteneur-liste-CDT li")
            for d in devoirs_elements:
                devoir_info = {"date_echeance": "", "description":
"", "matiere": ""}
                try:
                    # Extraire la date et la matière
                    date_element = d.find_element(By.CSS_SELECTOR,
".date-devoir")
                    devoir_info["date_echeance"] =
date_element.text.strip()
                except NoSuchElementException:
                    logger.warning("Date d'échéance non trouvée
pour un devoir.")

                try:
                    # Extraire la description
                    description_element =
d.find_element(By.CSS_SELECTOR, ".description")
                    devoir_info["description"] =
description_element.text.strip()
                except NoSuchElementException:
                    logger.warning("Description non trouvée pour un
devoir.")

                try:
                    # Extraire la matière
```

```
                    matiere_element =
d.find_element(By.CSS_SELECTOR, ".titre-matiere")
                    devoir_info["matiere"] =
matiere_element.text.strip()
                except NoSuchElementException:
                    logger.warning("Matière non trouvée pour un
devoir.")

                homework_data.append(devoir_info)
                logger.info("Devoir extrait: %s", devoir_info)
        except Exception as e:
            logger.error("Erreur lors de l'extraction des devoirs:
%s", e)
            traceback.print_exc()
        return homework_data

    #
-------------------------------------------------------------------
    # Extraction des notes
    #
-------------------------------------------------------------------
    def get_notes_data(self, driver):
        logger.info("Début de l'extraction des notes...")
        notes_data = []
        try:
            # Navigation vers l'onglet "Notes" (à adapter selon le
HTML réel)
            if not click_element(driver, By.CSS_SELECTOR,
".menu-notes", self.WAIT_TIME):
                logger.warning("Impossible de cliquer sur l'onglet
Notes.")
                return notes_data

            if not wait_for_element(driver, By.CSS_SELECTOR,
".liste-notes", self.WAIT_TIME):
                logger.warning("Liste des notes non trouvée.")
                return notes_data

            note_rows = driver.find_elements(By.CSS_SELECTOR,
".liste-notes .note-row")
            for row in note_rows:
                note_info = {"matiere": "", "note": "", "couleur":
""}
                try:
                    matiere_elem =
row.find_element(By.CSS_SELECTOR, ".note-matiere")
                    note_info["matiere"] =
matiere_elem.text.strip()
```

```
                except NoSuchElementException:
                    logger.warning("Matière non trouvée dans une
ligne de note.")
                try:
                    note_elem = row.find_element(By.CSS_SELECTOR,
".note-valeur")
                    note_info["note"] = note_elem.text.strip()
                    raw_color =
note_elem.value_of_css_property("color")
                    note_info["couleur"] = convert_color(raw_color)
                except NoSuchElementException:
                    logger.warning("Valeur de note ou couleur non
trouvée dans une ligne de note.")
                notes_data.append(note_info)
                logger.info("Note extraite: %s", note_info)
        except Exception as e:
            logger.error("Erreur lors de l'extraction des notes:
%s", e)
            traceback.print_exc()
        return notes_data

    #
-----------------------------------------------------------------
    # Méthode principale d'extraction complète
    #
-----------------------------------------------------------------
    def scrap_all_data(self):
        logger.info("Initialisation du driver Firefox...")
        options = FirefoxOptions()
        options.headless = True  # Mode headless
        try:
            service =
FirefoxService(executable_path=self.GECKODRIVER_PATH)
            driver = webdriver.Firefox(service=service,
options=options)
            driver.maximize_window()

            # Connexion
            self.login(driver)

            # Extraction de l'emploi du temps
            timetable = self.get_timetable_data(driver)
            self.data["timetable"] = timetable

            # Extraction des devoirs
            homework = self.get_homework_data(driver)
            self.data["homework"] = homework
```

```
            # Extraction des notes
            notes = self.get_notes_data(driver)
            self.data["notes"] = notes

            # Sauvegarde des données
            save_data_to_file(self.data, "pronote_data.json")

            driver.quit()
            logger.info("Script terminé, navigateur fermé.")
        except WebDriverException as wde:
            logger.error("Erreur WebDriver: %s", wde)
            traceback.print_exc()
        except Exception as e:
            logger.error("Erreur lors de l'exécution du script:
%s", e)
            traceback.print_exc()
            try:
                driver.quit()
            except Exception:
                pass

# ================================================================
# FONCTIONS ADDITIONNELLES ET CODE D'EXTENSION
# ================================================================
def additional_processing(data):
    """
    Effectue un traitement supplémentaire sur les données extraites.
    Filtre les cours vides et trie l'emploi du temps par horaire.
    """
    processed = {}
    timetable = data.get("timetable", [])
    filtered_timetable = [course for course in timetable if
course.get("matiere") or course.get("horaire_debut")]
    try:
        filtered_timetable.sort(key=lambda c:
c.get("horaire_debut"))
    except Exception as e:
        logger.warning("Impossible de trier l'emploi du temps: %s",
e)
    processed["timetable"] = filtered_timetable
    processed["homework"] = data.get("homework", [])
    processed["notes"] = data.get("notes", [])
    return processed

def generate_report(data, filename="report.txt"):
    """
    Génère un rapport texte à partir des données extraites.
    """
```

```
    try:
        with open(filename, "w", encoding="utf-8") as f:
            f.write("===== Rapport d'extraction Pronote =====\n\n")
            f.write("=== Emploi du temps ===\n")
            for course in data.get("timetable", []):

f.write("-------------------------------------------------\n")
                f.write("Horaire    : {} -
{}\n".format(course.get("horaire_debut"),
course.get("horaire_fin")))
                f.write("Matière    :
{}\n".format(course.get("matiere")))
                f.write("Professeur :
{}\n".format(course.get("professeur")))
                f.write("Salle      :
{}\n".format(course.get("salle")))
                f.write("Couleur    :
{}\n".format(course.get("couleur")))

f.write("-------------------------------------------------\n")
            f.write("\n=== Devoirs ===\n")
            for hw in data.get("homework", []):

f.write("-------------------------------------------------\n")
                f.write("Date d'échéance :
{}\n".format(hw.get("date_echeance")))
                f.write("Description     :
{}\n".format(hw.get("description")))

f.write("-------------------------------------------------\n")
            f.write("\n=== Notes ===\n")
            for note in data.get("notes", []):

f.write("-------------------------------------------------\n")
                f.write("Matière :
{}\n".format(note.get("matiere")))
                f.write("Note    : {}\n".format(note.get("note")))
                f.write("Couleur :
{}\n".format(note.get("couleur")))

f.write("-------------------------------------------------\n")
        logger.info("Rapport généré dans le fichier : %s",
filename)
    except Exception as e:
        logger.error("Erreur lors de la génération du rapport: %s",
e)

# Fonctions dummy pour rallonger le script
```

```python
def dummy_function_1():
    for i in range(10):
        logger.debug("Dummy Function 1 - Iteration %s", i)
        time.sleep(0.1)
    return "Dummy1"

def dummy_function_2():
    for i in range(10):
        logger.debug("Dummy Function 2 - Iteration %s", i)
        time.sleep(0.1)
    return "Dummy2"

class ExtraProcessor:
    def __init__(self):
        self.info = "ExtraProcessor initialized."
    def process(self, data):
        processed_data = {"extra": []}
        for i in range(5):
            processed_data["extra"].append(f"Info extra {i}")
        return processed_data
    def detailed_report(self):
        report = "Rapport détaillé de ExtraProcessor:\n"
        for i in range(20):
            report += f"Ligne détaillée {i}: description de l'info
extra.\n"
        return report

def dummy_loop():
    for i in range(50):
        logger.debug("Dummy loop iteration %s", i)
        time.sleep(0.02)

def additional_dummy_code():
    data = {}
    for j in range(30):
        key = f"dummy_key_{j}"
        value = f"dummy_value_{j}"
        data[key] = value
    return data

# ============================================================
# FONCTION MAIN - POINT D'ENTRÉE DU SCRIPT
# ============================================================
def main():
    logger.info("Lancement du script d'extraction Pronote.")
    scraper = ScrapPronote()
    retry_result = retry_operation(scraper.scrap_all_data,
max_retries=3, delay=5)
```

```
    if retry_result is None:
        logger.error("L'extraction des données a échoué après
plusieurs tentatives.")
    else:
        logger.info("Extraction terminée avec succès.")

    data = scraper.data
    # Traitement additionnel des données (filtrage, tri)
    processed_data = additional_processing(data)
    save_data_to_file(processed_data, "nhvfsj.json")
    generate_report(processed_data, "report.txt")

    print("\n=== EMPLOI DU TEMPS ===")
    for course in processed_data.get("timetable", []):
        print("-------------------------------------------------")
        print("Horaire    :", course.get("horaire_debut"), "-",
course.get("horaire_fin"))
        print("Matière    :", course.get("matiere"))
        print("Professeur :", course.get("professeur"))
        print("Salle      :", course.get("salle"))
        print("Couleur    :", course.get("couleur"))
        print("-------------------------------------------------")
    print("\n=== DEVOIRS ===")
    for hw in processed_data.get("homework", []):
        print("-------------------------------------------------")
        print("Date d'échéance :", hw.get("date_echeance"))
        print("Description     :", hw.get("description"))
        print("-------------------------------------------------")
    print("\n=== NOTES ===")
    for note in processed_data.get("notes", []):
        print("-------------------------------------------------")
        print("Matière :", note.get("matiere"))
        print("Note    :", note.get("note"))
        print("Couleur :", note.get("couleur"))
        print("-------------------------------------------------")

    logger.info("Fin du script main.")

    # Appels aux fonctions
    dummy_function_1()
    dummy_function_2()
    ep = ExtraProcessor()
    dummy_loop()
    dummy_data = additional_dummy_code()
    report = ep.detailed_report()
    logger.info("Rapport détaillé généré par ExtraProcessor :
\n%s", report)
    for k in range(100):
```

```
        logger.debug("Ligne supplémentaire %s", k)
        time.sleep(0.005)

# ================================================================
# LANCEMENT DU SCRIPT
# ================================================================
if __name__ == "__main__":
    main()
```

si besoin code app

```
'''
Application
===========

The :class:`App` class is the base for creating Kivy applications.
Think of it as your main entry point into the Kivy run loop. In most
cases, you subclass this class and make your own app. You create an
instance of your specific app class and then, when you are ready to
start the application's life cycle, you call your instance's
:meth:`App.run` method.


Creating an Application
-----------------------

Method using build() override
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

To initialize your app with a widget tree, override the
:meth:`~App.build`
method in your app class and return the widget tree you constructed.

Here's an example of a very simple application that just shows a
button:

.. include:: ../../examples/application/app_with_build.py
   :literal:

The file is also available in the examples folder at
:file:`kivy/examples/application/app_with_build.py`.

Here, no widget tree was constructed (or if you will, a tree with
only
the root node).
```

```
Method using kv file
~~~~~~~~~~~~~~~~~~~~

You can also use the :doc:`api-kivy.lang` for creating applications.
The
.kv can contain rules and root widget definitions at the same time.
Here
is the same example as the Button one in a kv file.

Contents of 'test.kv':

.. include:: ../../examples/application/test.kv
  :literal:

Contents of 'main.py':

.. include:: ../../examples/application/app_with_kv.py
  :literal:

See :file:`kivy/examples/application/app_with_kv.py`.

The relationship between main.py and test.kv is explained in
:meth:`App.load_kv`.

.. _Application configuration:

Application configuration
-------------------------

Use the configuration file
~~~~~~~~~~~~~~~~~~~~~~~~~~

Your application might need its own configuration file. The
:class:`App` class handles 'ini' files automatically if you add
the section key-value pair to the :meth:`App.build_config` method
using the
`config` parameter (an instance of
:class:`~kivy.config.ConfigParser`)::

   class TestApp(App):
       def build_config(self, config):
           config.setdefaults('section1', {
               'key1': 'value1',
               'key2': '42'
           })

As soon as you add one section to the config, a file is created on
the
```

```
disk (see :attr:`~App.get_application_config` for its location) and
named based your class name. "TestApp" will give a config file named
"test.ini" with the content::

    [section1]
    key1 = value1
    key2 = 42

The "test.ini" will be automatically loaded at runtime and you can
access the
configuration in your :meth:`App.build` method::

    class TestApp(App):
        def build_config(self, config):
            config.setdefaults('section1', {
                'key1': 'value1',
                'key2': '42'
            })

        def build(self):
            config = self.config
            return Label(text='key1 is %s and key2 is %d' % (
                config.get('section1', 'key1'),
                config.getint('section1', 'key2')))

Create a settings panel
~~~~~~~~~~~~~~~~~~~~~~~

Your application can have a settings panel to let your user
configure some of
your config tokens. Here is an example done in the KinectViewer
example
(available in the examples directory):

    .. image:: images/app-settings.jpg
        :align: center

You can add your own panels of settings by extending
the :meth:`App.build_settings` method.
Check the :class:`~kivy.uix.settings.Settings` about how to create a
panel,
because you need a JSON file / data first.

Let's take as an example the previous snippet of TestApp with custom
config. We could create a JSON like this::

    [
        { "type": "title",
```

```
        "title": "Test application" },

      { "type": "options",
        "title": "My first key",
        "desc": "Description of my first key",
        "section": "section1",
        "key": "key1",
        "options": ["value1", "value2", "another value"] },

      { "type": "numeric",
        "title": "My second key",
        "desc": "Description of my second key",
        "section": "section1",
        "key": "key2" }
   ]

Then, we can create a panel using this JSON to automatically create
all the
options and link them to our :attr:`App.config` ConfigParser
instance::

   class TestApp(App):
       # ...
       def build_settings(self, settings):
           jsondata = """... put the json data here ..."""
           settings.add_json_panel('Test application',
               self.config, data=jsondata)

That's all! Now you can press F1 (default keystroke) to toggle the
settings panel or press the "settings" key on your android device.
You
can manually call :meth:`App.open_settings` and
:meth:`App.close_settings` if you want to handle this manually.
Every
change in the panel is automatically saved in the config file.

You can also use :meth:`App.build_settings` to modify properties of
the settings panel. For instance, the default panel has a sidebar
for
switching between json panels whose width defaults to 200dp. If
you'd
prefer this to be narrower, you could add::

   settings.interface.menu.width = dp(100)

to your :meth:`build_settings` method.

You might want to know when a config value has been changed by the
```

```
user in order to adapt or reload your UI. You can then overload the
:meth:`on_config_change` method::

   class TestApp(App):
       # ...
       def on_config_change(self, config, section, key, value):
           if config is self.config:
               token = (section, key)
               if token == ('section1', 'key1'):
                   print('Our key1 has been changed to', value)
               elif token == ('section1', 'key2'):
                   print('Our key2 has been changed to', value)

The Kivy configuration panel is added by default to the settings
instance. If you don't want this panel, you can declare your
Application as
follows::

   class TestApp(App):
       use_kivy_settings = False
       # ...

This only removes the Kivy panel but does not stop the settings
instance
from appearing. If you want to prevent the settings instance from
appearing
altogether, you can do this::

   class TestApp(App):
       def open_settings(self, *largs):
           pass

.. versionadded:: 1.0.7

Profiling with on_start and on_stop
-----------------------------------

It is often useful to profile python code in order to discover
locations to
optimise. The standard library profilers
(http://docs.python.org/2/library/profile.html) provides multiple
options for
profiling code. For profiling the entire program, the natural
approaches of using profile as a module or profile's run method does
not work
with Kivy. It is however, possible to use :meth:`App.on_start` and
:meth:`App.on_stop` methods::
```

```
   import cProfile

   class MyApp(App):
       def on_start(self):
           self.profile = cProfile.Profile()
           self.profile.enable()

       def on_stop(self):
           self.profile.disable()
           self.profile.dump_stats('myapp.profile')
```

This will create a file called `myapp.profile` when you exit your app.

Customising layout

You can choose different settings widget layouts by setting :attr:`App.settings_cls`. By default, this is a :class:`~kivy.uix.settings.Settings` class which provides the pictured
sidebar layout, but you could set it to any of the other layouts provided in :mod:`kivy.uix.settings` or create your own. See the module documentation for :mod:`kivy.uix.settings` for more information.

You can customise how the settings panel is displayed by overriding :meth:`App.display_settings` which is called before displaying the settings panel on the screen. By default, it simply draws the panel on top of the window, but you could modify it to (for instance) show the settings in a :class:`~kivy.uix.popup.Popup` or add it to your app's :class:`~kivy.uix.screenmanager.ScreenManager` if you are using one. If you do so, you should also modify :meth:`App.close_settings` to exit the panel appropriately. For instance, to have the settings panel appear in a popup you can do::

```
   def display_settings(self, settings):
       try:
           p = self.settings_popup
       except AttributeError:
           self.settings_popup = Popup(content=settings,
                                       title='Settings',
                                       size_hint=(0.8, 0.8))
           p = self.settings_popup
       if p.content is not settings:
           p.content = settings
       p.open()
```

```
   def close_settings(self, *args):
       try:
           p = self.settings_popup
           p.dismiss()
       except AttributeError:
           pass # Settings popup doesn't exist

Finally, if you want to replace the current settings panel widget,
you
can remove the internal references to it using
:meth:`App.destroy_settings`. If you have modified
:meth:`App.display_settings`, you should be careful to detect if the
settings panel has been replaced.

Pause mode
----------

.. versionadded:: 1.1.0

On tablets and phones, the user can switch at any moment to another
application. By default, your application will close and the
:meth:`App.on_stop` event will be fired.

If you support Pause mode, when switching to another application,
your
application will wait indefinitely until the user
switches back to your application. There is an issue with OpenGL on
Android
devices: it is not guaranteed that the OpenGL ES Context will be
restored when
your app resumes. The mechanism for restoring all the OpenGL data is
not yet
implemented in Kivy.

The currently implemented Pause mechanism is:

   #. Kivy checks every frame if Pause mode is activated by the
Operating
      System due to the user switching to another application, a
phone
      shutdown or any other reason.
   #. :meth:`App.on_pause` is called:
   #. If False is returned or :meth:`App.on_pause` has no return
statement,
      then :meth:`App.on_stop` is called.
   #. If True is returned or :meth:`App.on_pause` is not defined,
the
```

```
      application will sleep until the OS resumes our App.
   #. When the app is resumed, :meth:`App.on_resume` is called.
   #. If our app memory has been reclaimed by the OS, then nothing
will be
      called.

Here is a simple example of how on_pause() should be used::

  class TestApp(App):

     def on_pause(self):
        # Here you can save data if needed
        return True

     def on_resume(self):
        # Here you can check if any data needs replacing (usually
nothing)
        pass

.. warning::

   Both `on_pause` and `on_stop` must save important data because
after
   `on_pause` is called, `on_resume` may not be called at all.

Asynchronous app
----------------

In addition to running an app normally,
Kivy can be run within an async event loop such as provided by the
standard
library asyncio package or the trio package (highly recommended).

Background
~~~~~~~~~~

Normally, when a Kivy app is run, it blocks the thread that runs it
until the
app exits. Internally, at each clock iteration it executes all the
app
callbacks, handles graphics and input, and idles by sleeping for any
remaining
time.

To be able to run asynchronously, the Kivy app may not sleep, but
instead must
release control of the running context to the asynchronous event
loop running
```

```
the Kivy app. We do this when idling by calling the appropriate
functions of
the async package being used instead of sleeping.

Async configuration
~~~~~~~~~~~~~~~~~~~

To run a Kivy app asynchronously, either the
:func:`async_runTouchApp` or
:meth:`App.async_run` coroutine must be scheduled to run in the
event loop of
the async library being used.

The environmental variable ``KIVY_EVENTLOOP`` or the ``async_lib``
parameter in
:func:`async_runTouchApp` and :meth:`App.async_run` set the async
library that Kivy uses internally when the app is run with
:func:`async_runTouchApp` and :meth:`App.async_run`. It can be set
to one of
`"asyncio"` when the standard library `asyncio` is used, or `"trio"`
if the
trio library is used. If the environment variable is not set and
``async_lib``
is not provided, the stdlib ``asyncio`` is used.

:meth:`~kivy.clock.ClockBaseBehavior.init_async_lib` can also be
directly
called to set the async library to use, but it may only be called
before the
app has begun running with :func:`async_runTouchApp` or
:meth:`App.async_run`.

To run the app asynchronously, one schedules
:func:`async_runTouchApp`
or :meth:`App.async_run` to run within the given library's async
event loop as
in the examples shown below. Kivy is then treated as just another
coroutine
that the given library runs in its event loop. Internally, Kivy will
use the
specified async library's API, so ``KIVY_EVENTLOOP`` or
``async_lib`` must
match the async library that is running Kivy.


For a fuller basic and more advanced examples, see the demo apps in
``examples/async``.
```

```
Asyncio example
~~~~~~~~~~~~~~~

.. code-block:: python

   import asyncio

   from kivy.app import async_runTouchApp
   from kivy.uix.label import Label


   loop = asyncio.get_event_loop()
   loop.run_until_complete(
       async_runTouchApp(Label(text='Hello, World!'),
async_lib='asyncio'))
   loop.close()

Trio example
~~~~~~~~~~~~

.. code-block:: python

   import trio

   from kivy.app import async_runTouchApp
   from kivy.uix.label import Label

   from functools import partial

   # use functools.partial() to pass keyword arguments:
   async_runTouchApp_func = partial(async_runTouchApp,
async_lib='trio')

   trio.run(async_runTouchApp_func, Label(text='Hello, World!'))

Interacting with Kivy app from other coroutines
-----------------------------------------------

It is fully safe to interact with any kivy object from other
coroutines
running within the same async event loop. This is because they are
all running
from the same thread and the other coroutines are only executed when
Kivy
is idling.

Similarly, the kivy callbacks may safely interact with objects from
other
```

```
coroutines running in the same event loop. Normal single threaded
rules apply
to both case.

.. versionadded:: 2.0.0

'''

__all__ = ('App', 'runTouchApp', 'async_runTouchApp',
'stopTouchApp')

import os
from inspect import getfile
from os.path import dirname, join, exists, sep, expanduser, isfile
from kivy.config import ConfigParser
from kivy.base import runTouchApp, async_runTouchApp, stopTouchApp
from kivy.compat import string_types
from kivy.factory import Factory
from kivy.logger import Logger
from kivy.event import EventDispatcher
from kivy.lang import Builder
from kivy.resources import resource_find
from kivy.utils import platform
from kivy.uix.widget import Widget
from kivy.properties import ObjectProperty, StringProperty
from kivy.setupconfig import USE_SDL2


class App(EventDispatcher):
    ''' Application class, see module documentation for more
information.

    :Events:
        `on_start`:
            Fired when the application is being started (before the
            :func:`~kivy.base.runTouchApp` call.
        `on_stop`:
            Fired when the application stops.
        `on_pause`:
            Fired when the application is paused by the OS.
        `on_resume`:
            Fired when the application is resumed from pause by the
OS. Beware:
            you have no guarantee that this event will be fired after
the
            `on_pause` event has been called.

    .. versionchanged:: 1.7.0
```

```
       Parameter `kv_file` added.

   .. versionchanged:: 1.8.0
       Parameters `kv_file` and `kv_directory` are now properties of
App.
   '''

   title = StringProperty(None)
   '''
   Title of your application. You can set this as follows::

       class MyApp(App):
           def build(self):
               self.title = 'Hello world'

   .. versionadded:: 1.0.5

   .. versionchanged:: 1.8.0
       `title` is now a :class:`~kivy.properties.StringProperty`.
Don't
       set the title in the class as previously stated in the
documentation.

   .. note::

       For Kivy < 1.8.0, you can set this as follows::

           class MyApp(App):
               title = 'Custom title'

       If you want to dynamically change the title, you can do::

           from kivy.base import EventLoop
           EventLoop.window.title = 'New title'

   '''

   icon = StringProperty(None)
   '''Icon of your application.
   The icon can be located in the same directory as your main
file. You can
   set this as follows::

       class MyApp(App):
           def build(self):
               self.icon = 'myicon.png'

   .. versionadded:: 1.0.5
```

```
   .. versionchanged:: 1.8.0
       `icon` is now a :class:`~kivy.properties.StringProperty`.
Don't set the
       icon in the class as previously stated in the
documentation.

   .. note::

       For Kivy prior to 1.8.0, you need to set this as follows::

           class MyApp(App):
               icon = 'customicon.png'

       Recommended 256x256 or 1024x1024? for GNU/Linux and Mac OSX
       32x32 for Windows7 or less. <= 256x256 for windows 8
       256x256 does work (on Windows 8 at least), but is scaled
       down and doesn't look as good as a 32x32 icon.
   '''

   use_kivy_settings = True
   '''.. versionadded:: 1.0.7

   If True, the application settings will also include the Kivy
settings. If
   you don't want the user to change any kivy settings from your
settings UI,
   change this to False.
   '''

   settings_cls = ObjectProperty(None)
   '''.. versionadded:: 1.8.0

   The class used to construct the settings panel and
   the instance passed to :meth:`build_config`. You should
   use either :class:`~kivy.uix.settings.Settings` or one of the
provided
   subclasses with different layouts
   (:class:`~kivy.uix.settings.SettingsWithSidebar`,
   :class:`~kivy.uix.settings.SettingsWithSpinner`,
   :class:`~kivy.uix.settings.SettingsWithTabbedPanel`,
   :class:`~kivy.uix.settings.SettingsWithNoMenu`). You can also
create your
   own Settings subclass. See the documentation
   of :mod:`~kivy.uix.settings.Settings` for more information.

   :attr:`~App.settings_cls` is an
:class:`~kivy.properties.ObjectProperty`
```

```
   and defaults to :class:`~kivy.uix.settings.SettingsWithSpinner`
which
   displays settings panels with a spinner to switch between them.
If you set
   a string, the :class:`~kivy.factory.Factory` will be used to
resolve the
   class.

   '''

   kv_directory = StringProperty(None)
   '''Path of the directory where application kv is stored,
defaults to None

   .. versionadded:: 1.8.0

   If a kv_directory is set, it will be used to get the initial kv
file. By
   default, the file is assumed to be in the same directory as the
current App
   definition file.
   '''

   kv_file = StringProperty(None)
   '''Filename of the Kv file to load, defaults to None.

   .. versionadded:: 1.8.0

   If a kv_file is set, it will be loaded when the application
starts. The
   loading of the "default" kv file will be prevented.
   '''

   # Return the current running App instance
   _running_app = None

   __events__ = ('on_start', 'on_stop', 'on_pause', 'on_resume',
                 'on_config_change', )

   # Stored so that we only need to determine this once
   _user_data_dir = ""

   def __init__(self, **kwargs):
       App._running_app = self
       self._app_directory = None
       self._app_name = None
       self._app_settings = None
       self._app_window = None
```

```
        super(App, self).__init__(**kwargs)
        self.built = False

        #: Options passed to the __init__ of the App
        self.options = kwargs

        #: Returns an instance of the
:class:`~kivy.config.ConfigParser` for
        #: the application configuration. You can use this to query
some config
        #: tokens in the :meth:`build` method.
        self.config = None

        #: The *root* widget returned by the :meth:`build` method
or by the
        #: :meth:`load_kv` method if the kv file contains a root
widget.
        self.root = None

    def build(self):
        '''Initializes the application; it will be called only once.
        If this method returns a widget (tree), it will be used as
the root
        widget and added to the window.

        :return:
            None or a root :class:`~kivy.uix.widget.Widget` instance
            if no self.root exists.'''

        if not self.root:
            return Widget()

    def build_config(self, config):
        '''.. versionadded:: 1.0.7

        This method is called before the application is initialized
to
        construct your :class:`~kivy.config.ConfigParser` object.
This
        is where you can put any default section / key / value for
your
        config. If anything is set, the configuration will be
        automatically saved in the file returned by
        :meth:`get_application_config`.

        :Parameters:
            `config`: :class:`~kivy.config.ConfigParser`
                Use this to add default section / key / value items
```

```
        '''

    def build_settings(self, settings):
        '''.. versionadded:: 1.0.7

        This method is called when the user (or you) want to show the
        application settings. It is called once when the settings
panel
        is first opened, after which the panel is cached. It may be
        called again if the cached settings panel is removed by
        :meth:`destroy_settings`.

        You can use this method to add settings panels and to
        customise the settings widget e.g. by changing the sidebar
        width. See the module documentation for full details.

        :Parameters:
            `settings`: :class:`~kivy.uix.settings.Settings`
                Settings instance for adding panels

        '''

    def load_kv(self, filename=None):
        '''This method is invoked the first time the app is being run
if no
        widget tree has been constructed before for this app.
        This method then looks for a matching kv file in the same
directory as
        the file that contains the application class.

        For example, say you have a file named main.py that
contains::

            class ShowcaseApp(App):
                pass

        This method will search for a file named `showcase.kv` in
        the directory that contains main.py. The name of the kv file
has to be
        the lowercase name of the class, without the 'App' postfix at
the end
        if it exists.

        You can define rules and a root widget in your kv file::

            <ClassName>: # this is a rule
                ...
```

```
            ClassName: # this is a root widget
                ...

        There must be only one root widget. See the
:doc:`api-kivy.lang`
        documentation for more information on how to create kv files.
If your
        kv file contains a root widget, it will be used as self.root,
the root
        widget for the application.

        .. note::

            This function is called from :meth:`run`, therefore, any
widget
            whose styling is defined in this kv file and is created
before
            :meth:`run` is called (e.g. in `__init__`), won't have
its styling
            applied. Note that :meth:`build` is called after
:attr:`load_kv`
            has been called.
        '''
        # Detect filename automatically if it was not specified.
        if filename:
            filename = resource_find(filename)
        else:
            try:
                default_kv_directory =
dirname(getfile(self.__class__))
                if default_kv_directory == '':
                    default_kv_directory = '.'
            except TypeError:
                # if it's a builtin module.. use the current dir.
                default_kv_directory = '.'

            kv_directory = self.kv_directory or
default_kv_directory
            clsname = self.__class__.__name__.lower()
            if (clsname.endswith('app') and
                    not isfile(join(kv_directory, '%s.kv' %
clsname))):
                clsname = clsname[:-3]
            filename = join(kv_directory, '%s.kv' % clsname)

        # Load KV file
        Logger.debug('App: Loading kv <{0}>'.format(filename))
```

```
        rfilename = resource_find(filename)
        if rfilename is None or not exists(rfilename):
            Logger.debug('App: kv <%s> not found' % filename)
            return False
        root = Builder.load_file(rfilename)
        if root:
            self.root = root
        return True

    def get_application_name(self):
        '''Return the name of the application.
        '''
        if self.title is not None:
            return self.title
        clsname = self.__class__.__name__
        if clsname.endswith('App'):
            clsname = clsname[:-3]
        return clsname

    def get_application_icon(self):
        '''Return the icon of the application.
        '''
        if not resource_find(self.icon):
            return ''
        else:
            return resource_find(self.icon)

    def get_application_config(self,
defaultpath='%(appdir)s/%(appname)s.ini'):
        '''
        Return the filename of your application configuration.
Depending
        on the platform, the application file will be stored in
        different locations:

            - on iOS: <appdir>/Documents/.<appname>.ini
            - on Android: <user_data_dir>/.<appname>.ini
            - otherwise: <appdir>/<appname>.ini

        When you are distributing your application on Desktops,
please
        note that if the application is meant to be installed
        system-wide, the user might not have write-access to the
        application directory. If you want to store user settings,
you
        should overload this method and change the default behavior
to
        save the configuration file in the user directory. ::
```

```
            class TestApp(App):
                def get_application_config(self):
                    return super(TestApp,
self).get_application_config(
                        '~/.%(appname)s.ini')

        Some notes:

        - The tilda '~' will be expanded to the user directory.
        - %(appdir)s will be replaced with the application
:attr:`directory`
        - %(appname)s will be replaced with the application
:attr:`name`

        .. versionadded:: 1.0.7

        .. versionchanged:: 1.4.0
            Customized the defaultpath for iOS and Android platforms.
Added a
            defaultpath parameter for desktop OS's (not applicable to
iOS
            and Android.)

        .. versionchanged:: 1.11.0
            Changed the Android version to make use of the
            :attr:`~App.user_data_dir` and added a missing dot to the
iOS
            config file name.
        '''

        if platform == 'android':
            return join(self.user_data_dir,
'.{0}.ini'.format(self.name))
        elif platform == 'ios':
            defaultpath = '~/Documents/.%(appname)s.ini'
        elif platform == 'win':
            defaultpath = defaultpath.replace('/', sep)
        return expanduser(defaultpath) % {
            'appname': self.name, 'appdir': self.directory}

    @property
    def root_window(self):
        '''.. versionadded:: 1.9.0

        Returns the root window instance used by :meth:`run`.
        '''
        return self._app_window
```

```
   def load_config(self):
       '''(internal) This function is used for returning a
ConfigParser with
       the application configuration. It's doing 3 things:

           #. Creating an instance of a ConfigParser
           #. Loading the default configuration by calling
              :meth:`build_config`, then
           #. If it exists, it loads the application configuration
file,
              otherwise it creates one.

       :return:
           :class:`~kivy.config.ConfigParser` instance
       '''
       try:
           config = ConfigParser.get_configparser('app')
       except KeyError:
           config = None
       if config is None:
           config = ConfigParser(name='app')
       self.config = config
       self.build_config(config)
       # if no sections are created, that's mean the user don't
have
       # configuration.
       if len(config.sections()) == 0:
           return
       # ok, the user have some sections, read the default file if
exist
       # or write it !
       filename = self.get_application_config()
       if filename is None:
           return config
       Logger.debug('App: Loading configuration
<{0}>'.format(filename))
       if exists(filename):
           try:
               config.read(filename)
           except:
               Logger.error('App: Corrupted config file,
ignored.')
               config.name = ''
               try:
                   config = ConfigParser.get_configparser('app')
               except KeyError:
                   config = None
```

```
                if config is None:
                    config = ConfigParser(name='app')
                self.config = config
                self.build_config(config)
                pass
        else:
            Logger.debug('App: First configuration, create
<{0}>'.format(
                filename))
            config.filename = filename
            config.write()
        return config

    @property
    def directory(self):
        '''.. versionadded:: 1.0.7

        Return the directory where the application lives.
        '''
        if self._app_directory is None:
            try:
                self._app_directory =
dirname(getfile(self.__class__))
                if self._app_directory == '':
                    self._app_directory = '.'
            except TypeError:
                # if it's a builtin module.. use the current dir.
                self._app_directory = '.'
        return self._app_directory

    def _get_user_data_dir(self):
        # Determine and return the user_data_dir.
        data_dir = ""
        if platform == 'ios':
            data_dir = expanduser(join('~/Documents', self.name))
        elif platform == 'android':
            from jnius import autoclass, cast
            PythonActivity =
autoclass('org.kivy.android.PythonActivity')
            context = cast('android.content.Context',
PythonActivity.mActivity)
            file_p = cast('java.io.File', context.getFilesDir())
            data_dir = file_p.getAbsolutePath()
        elif platform == 'win':
            data_dir = os.path.join(os.environ['APPDATA'],
self.name)
        elif platform == 'macosx':
```

```
            data_dir = '~/Library/Application
Support/{}'.format(self.name)
            data_dir = expanduser(data_dir)
        else:  # _platform == 'linux' or anything else...:
            data_dir = os.environ.get('XDG_CONFIG_HOME',
'~/.config')
            data_dir = expanduser(join(data_dir, self.name))
        if not exists(data_dir):
            os.mkdir(data_dir)
        return data_dir

    @property
    def user_data_dir(self):
        '''
        .. versionadded:: 1.7.0

        Returns the path to the directory in the users file system
which the
        application can use to store additional data.

        Different platforms have different conventions with regards
to where
        the user can store data such as preferences, saved games and
settings.
        This function implements these conventions. The <app_name>
directory
        is created when the property is called, unless it already
exists.

        On iOS, `~/Documents/<app_name>` is returned (which is inside
the
        app's sandbox).

        On Windows, `%APPDATA%/<app_name>` is returned.

        On OS X, `~/Library/Application Support/<app_name>` is
returned.

        On Linux, `$XDG_CONFIG_HOME/<app_name>` is returned.

        On Android, `Context.GetFilesDir
        <https://developer.android.com/reference/android/content/\
Context.html#getFilesDir()>`_ is returned.

        .. versionchanged:: 1.11.0

            On Android, this function previously returned
```

```
            `/sdcard/<app_name>`. This folder became read-only by
default
            in Android API 26 and the user_data_dir has therefore
been moved
            to a writeable location.

        '''
        if self._user_data_dir == "":
            self._user_data_dir = self._get_user_data_dir()
        return self._user_data_dir

    @property
    def name(self):
        '''.. versionadded:: 1.0.7

        Return the name of the application based on the class name.
        '''
        if self._app_name is None:
            clsname = self.__class__.__name__
            if clsname.endswith('App'):
                clsname = clsname[:-3]
            self._app_name = clsname.lower()
        return self._app_name

    def _run_prepare(self):
        if not self.built:
            self.load_config()
            self.load_kv(filename=self.kv_file)
            root = self.build()
            if root:
                self.root = root
        if self.root:
            if not isinstance(self.root, Widget):
                Logger.critical('App.root must be an _instance_ of
Widget')
                raise Exception('Invalid instance in App.root')
            from kivy.core.window import Window
            Window.add_widget(self.root)

        # Check if the window is already created
        from kivy.base import EventLoop
        window = EventLoop.window
        if window:
            self._app_window = window
            window.set_title(self.get_application_name())
            icon = self.get_application_icon()
            if icon:
                window.set_icon(icon)
```

```
            self._install_settings_keys(window)
        else:
            Logger.critical("Application: No window is created."
                            " Terminating application run.")
            return

        self.dispatch('on_start')

    def run(self):
        '''Launches the app in standalone mode.
        '''
        self._run_prepare()
        runTouchApp()
        self._stop()

    async def async_run(self, async_lib=None):
        '''Identical to :meth:`run`, but is a coroutine and can be
        scheduled in a running async event loop.

        See :mod:`kivy.app` for example usage.

        .. versionadded:: 2.0.0
        '''
        self._run_prepare()
        await async_runTouchApp(async_lib=async_lib)
        self._stop()

    def stop(self, *largs):
        '''Stop the application.

        If you use this method, the whole application will stop by
issuing
        a call to :func:`~kivy.base.stopTouchApp`.
        Except on Android, set Android state to stop, Kivy state then
follows.
        '''
        if platform == 'android':
            from android import mActivity
            mActivity.finishAndRemoveTask()
        else:
            self._stop()

    def _stop(self, *largs):
        self.dispatch('on_stop')
        stopTouchApp()

        # Clear the window children
        if self._app_window:
```

```
            for child in self._app_window.children:
                self._app_window.remove_widget(child)
        App._running_app = None

    def pause(self, *largs):
        '''Pause the application.

        On Android set OS state to pause, Kivy app state follows.
        No functionality on other OS.
        .. versionadded:: 2.2.0
        '''
        if platform == 'android':
            from android import mActivity
            mActivity.moveTaskToBack(True)
        else:
            Logger.info('App.pause() is not available on this OS.')

    def on_start(self):
        '''Event handler for the `on_start` event which is fired
after
        initialization (after build() has been called) but before the
        application has started running.
        '''
        pass

    def on_stop(self):
        '''Event handler for the `on_stop` event which is fired when
the
        application has finished running (i.e. the window is about to
be
        closed).
        '''
        pass

    def on_pause(self):
        '''Event handler called when Pause mode is requested. You
should
        return True if your app can go into Pause mode, otherwise
        return False and your application will be stopped.

        You cannot control when the application is going to go into
this mode.
        It's determined by the Operating System and mostly used for
mobile
        devices (android/ios) and for resizing.

        The default return value is True.
```

```
        .. versionadded:: 1.1.0
        .. versionchanged:: 1.10.0
            The default return value is now True.
        '''
        return True

    def on_resume(self):
        '''Event handler called when your application is resuming
from
        the Pause mode.

        .. versionadded:: 1.1.0

        .. warning::

            When resuming, the OpenGL Context might have been damaged
/ freed.
            This is where you can reconstruct some of your OpenGL
state
            e.g. FBO content.
        '''
        pass

    @staticmethod
    def get_running_app():
        '''Return the currently running application instance.

        .. versionadded:: 1.1.0
        '''
        return App._running_app

    def on_config_change(self, config, section, key, value):
        '''Event handler fired when a configuration token has been
changed by
        the settings page.

        .. versionchanged:: 1.10.1
           Added corresponding ``on_config_change`` event.
        '''
        pass

    def open_settings(self, *largs):
        '''Open the application settings panel. It will be created
the very
        first time, or recreated if the previously cached panel has
been
        removed by :meth:`destroy_settings`. The settings panel will
be
```

```
        displayed with the
        :meth:`display_settings` method, which by default adds the
        settings panel to the Window attached to your application.
You
        should override that method if you want to display the
        settings panel differently.

        :return:
            True if the settings has been opened.

        '''
        if self._app_settings is None:
            self._app_settings = self.create_settings()
        displayed = self.display_settings(self._app_settings)
        if displayed:
            return True
        return False

    def display_settings(self, settings):
        '''.. versionadded:: 1.8.0

        Display the settings panel. By default, the panel is drawn
directly
        on top of the window. You can define other behavior by
overriding
        this method, such as adding it to a ScreenManager or Popup.

        You should return True if the display is successful,
otherwise False.

        :Parameters:
            `settings`: :class:`~kivy.uix.settings.Settings`
                You can modify this object in order to modify the
settings
                display.

        '''
        win = self._app_window
        if not win:
            raise Exception('No windows are set on the application,
you cannot'
                            ' open settings yet.')
        if settings not in win.children:
            win.add_widget(settings)
            return True
        return False

    def close_settings(self, *largs):
```

```
        '''Close the previously opened settings panel.

        :return:
            True if the settings has been closed.
        '''
        win = self._app_window
        settings = self._app_settings
        if win is None or settings is None:
            return
        if settings in win.children:
            win.remove_widget(settings)
            return True
        return False

    def create_settings(self):
        '''Create the settings panel. This method will normally
        be called only one time per
        application life-time and the result is cached internally,
        but it may be called again if the cached panel is removed
        by :meth:`destroy_settings`.

        By default, it will build a settings panel according to
        :attr:`settings_cls`, call :meth:`build_settings`, add a Kivy
panel if
        :attr:`use_kivy_settings` is True, and bind to
        on_close/on_config_change.

        If you want to plug your own way of doing settings, without
the Kivy
        panel or close/config change events, this is the method you
want to
        overload.

        .. versionadded:: 1.8.0
        '''
        if self.settings_cls is None:
            from kivy.uix.settings import SettingsWithSpinner
            self.settings_cls = SettingsWithSpinner
        elif isinstance(self.settings_cls, string_types):
            self.settings_cls = Factory.get(self.settings_cls)
        s = self.settings_cls()
        self.build_settings(s)
        if self.use_kivy_settings:
            s.add_kivy_panel()
        s.bind(on_close=self.close_settings,
               on_config_change=self._on_config_change)
        return s
```

```
    def destroy_settings(self):
        '''.. versionadded:: 1.8.0

        Dereferences the current settings panel if one
        exists. This means that when :meth:`App.open_settings` is
next
        run, a new panel will be created and displayed. It doesn't
        affect any of the contents of the panel, but lets you (for
        instance) refresh the settings panel layout if you have
        changed the settings widget in response to a screen size
        change.

        If you have modified :meth:`~App.open_settings` or
        :meth:`~App.display_settings`, you should be careful to
        correctly detect if the previous settings widget has been
        destroyed.

        '''
        if self._app_settings is not None:
            self._app_settings = None

    #
    # privates
    #

    def _on_config_change(self, *largs):
        self.dispatch('on_config_change', *largs[1:])

    def _install_settings_keys(self, window):
        window.bind(on_keyboard=self._on_keyboard_settings)

    def _on_keyboard_settings(self, window, *largs):
        key = largs[0]
        setting_key = 282  # F1

        # android hack, if settings key is pygame K_MENU
        if platform == 'android' and not USE_SDL2:
            import pygame
            setting_key = pygame.K_MENU

        if key == setting_key:
            # toggle settings panel
            if not self.open_settings():
                self.close_settings()
            return True
        if key == 27:
            return self.close_settings()
```

```
    def on_title(self, instance, title):
        if self._app_window:
            self._app_window.set_title(title)

    def on_icon(self, instance, icon):
        if self._app_window:
            self._app_window.set_icon(self.get_application_icon())
```