```python
# smartmirror.py
# requirements
# requests, feedparser, traceback, Pillow

from Tkinter import *
import locale
import threading
import time
import requests
import json
import traceback
import feedparser

from PIL import Image, ImageTk
from contextlib import contextmanager

LOCALE_LOCK = threading.Lock()

ip = '197.49.129.201'
ui_locale = '' # e.g. 'fr_FR' fro French, '' as default
time_format = 12 # 12 or 24
date_format = "%b %d, %Y" # check python doc for strftime() for options
news_country_code = 'EG'
weather_api_token = "3ca72348dd0be3026126487e3419b240" # create account at
https://darksky.net/dev/
weather_lang = 'en' # see https://darksky.net/dev/docs/forecast for full list of language
parameters values
weather_unit = 'si' # see https://darksky.net/dev/docs/forecast for full list of unit
parameters values
latitude =  '30.06263'# Set this if IP location lookup does not work for you (must be a string)
longitude = '31.24967' # Set this if IP location lookup does not work for you (must be a
string)
xlarge_text_size = 94
large_text_size = 48
medium_text_size = 28
small_text_size = 18

@contextmanager
def setlocale(name): #thread proof function to work with locale
    with LOCALE_LOCK:
        saved = locale.setlocale(locale.LC_ALL)
        try:
            yield locale.setlocale(locale.LC_ALL, name)
        finally:
            locale.setlocale(locale.LC_ALL, saved)

# maps open weather icons to
# icon reading is not impacted by the 'lang' parameter
icon_lookup = {
    'clear-day': "assets/Sun.png",  # clear sky day
    'wind': "assets/Wind.png",   #wind
    'cloudy': "assets/Cloud.png",  # cloudy day
    'partly-cloudy-day': "assets/PartlySunny.png",  # partly cloudy day
    'rain': "assets/Rain.png",  # rain day
    'snow': "assets/Snow.png",  # snow day
    'snow-thin': "assets/Snow.png",  # sleet day
    'fog': "assets/Haze.png",  # fog day
    'clear-night': "assets/Moon.png",  # clear sky night
    'partly-cloudy-night': "assets/PartlyMoon.png",  # scattered clouds night
    'thunderstorm': "assets/Storm.png",  # thunderstorm
    'tornado': "assests/Tornado.png",    # tornado
    'hail': "assests/Hail.png"  # hail
}
```

```
class Clock(Frame):
    def __init__(self, parent, *args, **kwargs):
        Frame.__init__(self, parent, bg='black')


        # initialize time label
        self.time1 = ''
        self.timeLbl = Label(self, font=('Helvetica', large_text_size), fg="white", bg="black")
        self.timeLbl.pack(side=TOP, anchor=E)
        # initialize day of week
        self.day_of_week1 = ''
        self.dayOWLbl = Label(self, text=self.day_of_week1, font=('Helvetica',
small_text_size), fg="white", bg="black")
        self.dayOWLbl.pack(side=TOP, anchor=E)
        # initialize date label
        self.date1 = ''
        self.dateLbl = Label(self, text=self.date1, font=('Helvetica', small_text_size),
fg="white", bg="black")
        self.dateLbl.pack(side=TOP, anchor=E)
        self.tick()

    def tick(self):
        with setlocale(ui_locale):
            if time_format == 12:
                time2 = time.strftime('%I:%M %p') #hour in 12h format
            else:
                time2 = time.strftime('%H:%M') #hour in 24h format

            day_of_week2 = time.strftime('%A')
            date2 = time.strftime(date_format)
            # if time string has changed, update it
            if time2 != self.time1:
                self.time1 = time2
                self.timeLbl.config(text=time2)
            if day_of_week2 != self.day_of_week1:
                self.day_of_week1 = day_of_week2
                self.dayOWLbl.config(text=day_of_week2)
            if date2 != self.date1:
                self.date1 = date2
                self.dateLbl.config(text=date2)
            # calls itself every 200 milliseconds
            # to update the time display as needed
            # could use >200 ms, but display gets jerky
            self.timeLbl.after(200, self.tick)
```

```
class Weather(Frame):
    def __init__(self, parent, *args, **kwargs):
        Frame.__init__(self, parent, bg='black')
        self.temperature = ''
        self.forecast = ''
        self.location = ''
        self.currently = ''
        self.icon = ''
        self.degreeFrm = Frame(self, bg="black")
        self.degreeFrm.pack(side=TOP, anchor=W)
        self.temperatureLbl = Label(self.degreeFrm, font=('Helvetica', xlarge_text_size),
fg="white", bg="black")
        self.temperatureLbl.pack(side=LEFT, anchor=N)
        self.iconLbl = Label(self.degreeFrm, bg="black")
        self.iconLbl.pack(side=LEFT, anchor=N, padx=20)
        self.currentlyLbl = Label(self, font=('Helvetica', medium_text_size), fg="white",
bg="black")
        self.currentlyLbl.pack(side=TOP, anchor=W)
        self.forecastLbl = Label(self, font=('Helvetica', small_text_size), fg="white",
bg="black")
        self.forecastLbl.pack(side=TOP, anchor=W)
        self.locationLbl = Label(self, font=('Helvetica', small_text_size), fg="white",
bg="black")
        self.locationLbl.pack(side=TOP, anchor=W)
        self.get_weather()

    def get_ip(self):
        try:
            ip_url = "http://jsonip.com/"
            req = requests.get(ip_url)
            ip_json = json.loads(req.text)
            return ip_json['ip']
        except Exception as e:
            traceback.print_exc()
            return "Error: %s. Cannot get ip." % e

    def get_weather(self):
        try:

            if latitude is None and longitude is None:
                # get location
                location_req_url = "http://freegeoip.net/json/%s" % self.get_ip()
                r = requests.get(location_req_url)
                location_obj = json.loads(r.text)

                lat = location_obj['latitude']
                lon = location_obj['longitude']

                location2 = "%s, %s" % (location_obj['city'], location_obj['region_code'])

                # get weather

                weather_req_url = "https://api.darksky.net/forecast/%s/%s,%s?lang=%s&units=%s"
% (weather_api_token, lat,lon,weather_lang,weather_unit)
            else:
                location2 = ""
                # get weather
                weather_req_url = "https://api.darksky.net/forecast/%s/%s,%s?lang=%s&units=%s"
% (weather_api_token, latitude, longitude, weather_lang, weather_unit)

            r = requests.get(weather_req_url)
            weather_obj = json.loads(r.text)

            degree_sign= u'\N{DEGREE SIGN}'
            temperature2 = "%s%s" % (str(int(weather_obj['currently']['temperature'])),
degree_sign)
            currently2 = weather_obj['currently']['summary']
            forecast2 = weather_obj["hourly"]["summary"]

            icon_id = weather_obj['currently']['icon']
            icon2 = None

            if icon_id in icon_lookup:
                icon2 = icon_lookup[icon_id]

            if icon2 is not None:
                if self.icon != icon2:
                    self.icon = icon2
                    image = Image.open(icon2)
                    image = image.resize((100, 100), Image.ANTIALIAS)
                    image = image.convert('RGB')
                    photo = ImageTk.PhotoImage(image)

                    self.iconLbl.config(image=photo)
                    self.iconLbl.image = photo
            else:
                # remove image
                self.iconLbl.config(image='')

            if self.currently != currently2:
                self.currently = currently2
                self.currentlyLbl.config(text=currently2)
            if self.forecast != forecast2:
                self.forecast = forecast2
                self.forecastLbl.config(text=forecast2)
            if self.temperature != temperature2:
                self.temperature = temperature2
                self.temperatureLbl.config(text=temperature2)
            if self.location != location2:
                if location2 == ", ":
                    self.location = "Cannot Pinpoint Location"
                    self.locationLbl.config(text="Cannot Pinpoint Location")
                else:
                    self.location = location2
                    self.locationLbl.config(text=location2)
        except Exception as e:
            traceback.print_exc()
            print "Error: %s. Cannot get weather." % e

        self.after(600000, self.get_weather)

    @staticmethod
    def convert_kelvin_to_fahrenheit(kelvin_temp):
        return 1.8 * (kelvin_temp - 273) + 32
```

```python
class News(Frame):
    def __init__(self, parent, *args, **kwargs):
        Frame.__init__(self, parent, *args, **kwargs)
        self.config(bg='black')


        self.title = 'News' # 'News' is more internationally generic
        self.newsLbl = Label(self, text=self.title, font=('Helvetica', medium_text_size),
fg="white", bg="black")
        self.newsLbl.pack(side=TOP, anchor=W)
        self.headlinesContainer = Frame(self, bg="black")
        self.headlinesContainer.pack(side=TOP)
        self.get_headlines()

    def get_headlines(self):
        try:
            # remove all children
            for widget in self.headlinesContainer.winfo_children():
                widget.destroy()
            if news_country_code == None:
                headlines_url = "https://news.google.com/news?ned=us&output=rss"
            else:
                headlines_url = "https://news.google.com/news?ned=%s&output=rss" %
news_country_code

            feed = feedparser.parse(headlines_url)

            for post in feed.entries[0:5]:
                headline = NewsHeadline(self.headlinesContainer, post.title)
                headline.pack(side=TOP, anchor=W)
        except Exception as e:
            traceback.print_exc()
            print "Error: %s. Cannot get news." % e

        self.after(600000, self.get_headlines)
```

```
class NewsHeadline(Frame):
    def __init__(self, parent, event_name=""):
        Frame.__init__(self, parent, bg='black')

        image = Image.open("assets/Newspaper.png")
        image = image.resize((25, 25), Image.ANTIALIAS)
        image = image.convert('RGB')
        photo = ImageTk.PhotoImage(image)

        self.iconLbl = Label(self, bg='black', image=photo)
        self.iconLbl.image = photo
        self.iconLbl.pack(side=LEFT, anchor=N)

        self.eventName = event_name
        self.eventNameLbl = Label(self, text=self.eventName, font=('Helvetica',
small_text_size), fg="white", bg="black")
        self.eventNameLbl.pack(side=LEFT, anchor=N)
```

```
class Calendar(Frame):
    def __init__(self, parent, *args, **kwargs):
        Frame.__init__(self, parent, bg='black')
        self.title = 'Calendar Events'
        self.calendarLbl = Label(self, text=self.title, font=('Helvetica', medium_text_size),
fg="white", bg="black")
        self.calendarLbl.pack(side=TOP, anchor=E)
        self.calendarEventContainer = Frame(self, bg='black')
        self.calendarEventContainer.pack(side=TOP, anchor=E)
        self.get_events()

    def get_events(self):
        #TODO: implement this method
        # reference https://developers.google.com/google-apps/calendar/quickstart/python

        # remove all children
        for widget in self.calendarEventContainer.winfo_children():
            widget.destroy()

        calendar_event = CalendarEvent(self.calendarEventContainer)
        calendar_event.pack(side=TOP, anchor=E)
        pass
```

```
class CalendarEvent(Frame):
    def __init__(self, parent, event_name="Event 1"):
        Frame.__init__(self, parent, bg='black')
        self.eventName = event_name
        self.eventNameLbl = Label(self, text=self.eventName, font=('Helvetica', 
small_text_size), fg="white", bg="black")
        self.eventNameLbl.pack(side=TOP, anchor=E)
```

```python
class Abdo(Frame):
    def __init__(self, parent, event_name="Event 1"):
        Frame.__init__(self, parent, bg='black')
       # self.a=u'\u0627\u0644\u0633\u0644\u0627\u0645 \u0639\u0644\u064a\u0643\u0645'

        #encoded=self.a.encode('utf-8')
        self.a ='Ø§Ù„Ø³Ù„Ø§Ù… '
        self.txt = Label(self, text=self.a, font=('Helvetica',44), fg="white", bg="black")
        self.txt.pack(side=TOP, anchor=E)
```

```python
class FullscreenWindow:

    def __init__(self):
        self.tk = Tk()
        self.tk.configure(background='black')
        self.topFrame = Frame(self.tk, background = 'black')
        self.bottomFrame = Frame(self.tk, background = 'black')
        self.topFrame.pack(side = TOP, fill=BOTH, expand = YES)
        self.bottomFrame.pack(side = BOTTOM, fill=BOTH, expand = YES)
        self.state = False
        self.tk.bind("<Return>", self.toggle_fullscreen)
        self.tk.bind("<Escape>", self.end_fullscreen)


        #text
        self.abdo =Abdo(self.topFrame)
        self.abdo.pack(side=TOP, anchor=N, padx=100, pady=60)

        # clock
        self.clock = Clock(self.topFrame)
        self.clock.pack(side=RIGHT, anchor=N, padx=100, pady=60)
        # weather
        self.weather = Weather(self.topFrame)
        self.weather.pack(side=LEFT, anchor=N, padx=100, pady=60)
        # news
        self.news = News(self.bottomFrame)
        self.news.pack(side=LEFT, anchor=S, padx=100, pady=60)
        # calender - removing for now
        # self.calender = Calendar(self.bottomFrame)
        # self.calender.pack(side = RIGHT, anchor=S, padx=100, pady=60)

    def toggle_fullscreen(self, event=None):
        self.state = not self.state  # Just toggling the boolean
        self.tk.attributes("-fullscreen", self.state)
        return "break"

    def end_fullscreen(self, event=None):
        self.state = False
        self.tk.attributes("-fullscreen", False)
        return "break"

if __name__ == '__main__':
    w = FullscreenWindow()
    w.tk.mainloop()
```