Lab1

任务说明

1. 本次任务完成的项目启发来自我上次去上海迪士尼排队时间过长，不同的乐园需要下载不同的APP，官方APP打开响应速度慢，没有一些自定义的功能。所以本次python课程作业想要完成一个能够包含大部分乐园，并且排队时间显示明确，用户可以自定义设置一个期望的排队时间，软件会在排队时间低于用户设置的时间后提醒用户。
2. 我们的软件包含如下功能：
    - 在列表内按照字典顺序显示所支持的所有乐园，并且这个乐园可以根据API的支持情况动态更新
    - 选择好乐园后在主视图中显示乐园的开园时间和闭园时间，以及每个项目的实时排队情况
    - 用户可以选择特定的游玩项目设置期望的排队时间，当实际排队时间低于期望排队时间后，程序会立刻提醒用户
    - 针对不同的排队时间，显示不同的颜色，便于用户感受排队时长
    - 本次实验我们尽可能做到完善的错误处理机制，我们尽量能够使得程序处理更多的错误情况

实验细节

API

- 本次试验的API来自于免费维护的ThemeParks.wiki
- 我们主要用到该API的`/destinations`方法，`/entity/{entityID}/live`方法以及`/entity/{entityID}/schedule`方法
    - `/destinations`方法可以得到该API支持的所有乐园情况并且获得对应乐园名称的`entityID`
    - `/entity/{entityID}/live`方法能够获得特定乐园的所有项目的排队情况
    - `/entity/{entityID}/schedule`方法能够获得该乐园的开园时间和闭园时间
- 我们先使用API的`/destinations`方法获得支持的乐园名称，然后在用户选择特定的乐园后，我们再获取该乐园支持的项目，以及项目所需要的排队时间，并且使用`/entity/{entityID}/schedule`方法提供乐园的开园和闭园时间供用户参考

GUI

- 本次实验为了优化GUI的显示效果，使用了第三方库customtkinter，该库在大多数情况下的使用方法与tkinter相同
- 本次实验的GUI主要设计`APP`的主类，并且为了程序的可读性，我们每创建一个新的`Frame`都会创建一个新的类来支持他，以便于程序的管理和可读性
    - `APP`类，该类是对外提供接口的类，该类创建了本项目最主要的GUI界面，并且将APP分为两大区域，分别为`park_list`区域和`ride_frame`区域，其中第一个区域提供所支持的所有公园，第二个区域显示所选定的公园的各个项目的排队时间

```
class App(customtkinter.CTk):
    parks = None

    def __init__(self):
        super().__init__()

        self.closed = None
        self.attractions = None
        customtkinter.set_appearance_mode("light")
        customtkinter.set_default_color_theme("dark-blue")

        self.title("Parks Queue Times")
        self.geometry("1920x1080")
        self.rowconfigure(0, weight=1)
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=15)

        self.park_frame = ParkFrame(self)
        self.park_frame.grid(row=0, column=0, sticky="nsew")
        self.ride_frame = RideFrame(self)
        self.ride_frame.grid(row=0, column=1, sticky="nsew")
        self.update()
        self.api = GETAPI.API()
        self.update_parks()

    def update_parks(self):
        self.parks = self.api.get_parks()
        if self.parks == 0:
            self.park_frame.get_list_error()
        else:
            self.park_frame.update_parks(self.parks)

    def update_rides(self, name):
        self.attractions, self.closed = self.api.get_park(name)
        if self.attractions == 0:
            self.ride_frame.get_ride_error(name)
        else:
            closing_time, opening_time = self.api.get_schedule(name)
            if closing_time == 0:
                self.ride_frame.get_ride_error(name)
            else:
                self.ride_frame.update_rides(name, self.attractions, self.closed,
closing_time, opening_time)
        self.update()
```

- `ParkFrame`该类提供了展示公园列表的各项功能

```
class ParkFrame(customtkinter.CTkFrame):
    def __init__(self, master):
        super().__init__(master)

        self.master = master
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(0, weight=1)
        self.grid_rowconfigure(1, weight=20)

        self.title = customtkinter.CTkLabel(self, text="Park Name",
fg_color="transparent", font=("Arial", 28))
        self.title.grid(row=0, column=0)

        self.park_list = ParkListFrame(self)
        self.park_list.grid(row=1, column=0, sticky="nsew")

    def update_parks(self, parks):
        self.title.configure(text="Park List")
        self.park_list.update_parks(parks)

    def get_list_error(self):
        self.park_list.get_list_error()

    def update_rides(self, attractions):
        self.title.configure(text="Ride List")
        self.park_list.update_rides(attractions)
```

- `RideFrame`类提供了展示项目的各个游玩时间的各种方法，以及相应的错误处理

```
class RideFrame(customtkinter.CTkFrame):
    def __init__(self, master):
        super().__init__(master)

        self.queue_time_frame = None
        self.master = master
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(0, weight=1)
        self.grid_rowconfigure(1, weight=1)
        self.grid_rowconfigure(2, weight=10)

        self.title = customtkinter.CTkLabel(self, text="Please choose the park
from the left list!",
                                            fg_color="transparent",
```

```
                                            font=("Arial", 28))
        self.title.grid(row=0, column=0, sticky="nsew")
        self.schedule = customtkinter.CTkLabel(self, text=" ",
                                               fg_color="transparent",
                                               text_color="grey",
                                               font=("Robot", 20))
        self.schedule.grid(row=1, column=0, sticky="nsew")
        self.queue_time_frame = QueueTimeFrame(self)
        self.queue_time_frame.grid(row=2, column=0, sticky="new")

    def update_rides(self, park_name, attractions, closed, closing_time,
opening_time):
        self.title.configure(text=park_name)
        self.schedule.configure(text="Opening time: " + opening_time + "\nClosing
time: " + closing_time)
        self.queue_time_frame.destroy()
        self.queue_time_frame = QueueTimeFrame(self)
        self.queue_time_frame.grid(row=2, column=0, sticky="nsew")
        self.queue_time_frame.update_ride(attractions, closed, park_name)
        self.update()

    def update_project(self, park_name, ride_name, wait_time):
        self.title.configure(text=ride_name)
        self.schedule.configure(text="Wait time: " + str(wait_time) + " minutes")
        self.queue_time_frame.destroy()
        self.queue_time_frame = ProjectFrame(self, park_name, ride_name)
        self.queue_time_frame.grid(row=2, column=0, sticky="nsew")

    def get_ride_error(self, name):
        self.queue_time_frame.destroy()
        self.title.configure(text=name)
        self.queue_time_frame = QueueTimeFrame(self)
        self.queue_time_frame.grid(row=2, column=0, sticky="nsew")
        self.queue_time_frame.get_ride_error(name)
        self.update()
```

- `TopLevelReminder`类，这个类可以创建一个新的窗口，提醒用户的排队时间

```
class TopLevelReminder(customtkinter.CTkToplevel):
    def __init__(self, master, park_name, ride_name):
        super().__init__(master)
        self.label = None
        self.park_name = park_name
        self.ride_name = ride_name
        self.geometry("400x300")
```

```
        self.title("Reminder")
        self.reminder_Label = customtkinter.CTkLabel(self,
                                                     text=ride_name,
                                                     fg_color="transparent",
                                                     text_color="black",
                                                     font=("Arial", 22))
        self.reminder_Label.grid(row=0, column=0, sticky="nsew")
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure((0, 1, 2, 3), weight=1)
        self.Label = customtkinter.CTkLabel(self,
                                            text="Remind me when the wait time
less than ",
                                            fg_color="transparent",
                                            text_color="black",
                                            font=("Arial", 18))
        self.Label.grid(row=1, column=0, sticky="nsew")
        self.option_menu = customtkinter.CTkOptionMenu(self,
                                                       values=["15min", "30min",
"45min", "60min", "75min", "90min"])
        self.option_menu.set("15min")
        self.option_menu.grid(row=2, column=0, padx=(30, 30), pady=(30, 30),
sticky="nsew")
        self.reminder_button = customtkinter.CTkButton(self, text="Remind me!",
                                                       command=self.remind_me,
                                                       font=("Robot", 18))
        self.reminder_button.grid(row=3, column=0, padx=(10, 10), pady=(10, 10),
sticky="nsew")

    def remind_me(self):
        time = self.option_menu.get()
        self.reminder_button.destroy()
        self.option_menu.destroy()
        self.label = customtkinter.CTkLabel(self, text_color="black",
                                            fg_color="transparent",
                                            text="Reminder set! ",
                                            font=("Arial", 18))
        self.label.grid(row=1, column=0, sticky="nsew")
        self.grid_rowconfigure(1, weight=1)
        self.update()
        p = Process(target=set_reminder, args=(self.park_name, self.ride_name,
int(time.removesuffix("min"))))
        p.start()
```

- `GUI`还有很多的类来辅助上面四个类实现具体的任务，比较繁琐并且简单易懂，如果想了解深入可以阅读源代码，我们细心地为读者添加了必要的注释

GETAPI

该部分主要讲述的是我们创建了一个名为`API`的类，用于完成请求`API`的各项功能，代码中有关于各个方法的介绍，并且较为浅显易懂，我们把代码附在下方，就不再赘述

```
class API:
    def __init__(self):
        self.api = "https://api.themeparks.wiki/v1"
        self.parks = {}
        self.closed = []
        self.attractions = {}

    def get_parks(self):
        """
        Returns a dictionary of parks
        :return:
        """
        try:
            parks_list_response = requests.get(self.api + "/destinations")
        except requests.exceptions.RequestException:
            return 0
        parks_list = json.loads(parks_list_response.text)
        for park_class in parks_list["destinations"]:
            for park in park_class["parks"]:
                self.parks[park["name"]] = park["id"]
        self.parks = dict(sorted(self.parks.items()))
        return self.parks

    def get_park(self, park_name):
        """
        Returns a dictionary of attractions and their wait times
        :param park_name:
        :return: attractions, closed
        """
        self.attractions = {}
        self.closed = []
        try:
            # print(self.api + "/entity/" + self.parks[park_name] + '/live')
            park_response = requests.get(self.api + "/entity/" + self.parks[park_name] +
'/live')
        except requests.exceptions.RequestException:
            return 0, 0
        park = json.loads(park_response.text)
        for live in park["liveData"]:
            if live["status"] == "CLOSED":
```

```
                self.closed.append(live["name"])
            elif live["entityType"] == "ATTRACTION":
                try:
                    if live["queue"]["STANDBY"]["waitTime"] is not None and live["status"]
== "OPERATING":
                        self.attractions[live["name"]] = live["queue"]["STANDBY"]
["waitTime"]
                except KeyError:
                    pass

        # sort attractions by wait time from highest to lowest
        self.attractions = dict(sorted(self.attractions.items(), key=lambda item: item[1],
reverse=True))

        return self.attractions, self.closed

    def get_schedule(self, park_name):
        """
        Returns the closing and opening time of the park
        """
        try:
            # print(self.api + '/entity/' + self.parks[park_name] + '/schedule')
            schedule_response = requests.get(self.api + '/entity/' + self.parks[park_name]
+ '/schedule')
        except requests.exceptions.RequestException:
            return 0, 0
        schedule = json.loads(schedule_response.text)
        return schedule["schedule"][1]["closingTime"], schedule["schedule"][1]
["openingTime"]

    def get_ride_time(self, park_name, ride_name):
        """
        Returns the wait time of a ride
        """
        try:
            park_response = requests.get(self.api + "/entity/" + self.parks[park_name] +
'/live')
        except requests.exceptions.RequestException:
            return -1
        park = json.loads(park_response.text)
        try:
            for live in park["liveData"]:
                if live["name"] == ride_name:
                    return live["queue"]["STANDBY"]["waitTime"]
        except KeyError:
            return -1
```

MAIN

`main`函数只需要调用GUI的相关功能即可

```
import GUI


# Init Graphical User Interface
if __name__ == "__main__":
    app = GUI.App()
    app.mainloop()
```

实验中遇到的问题

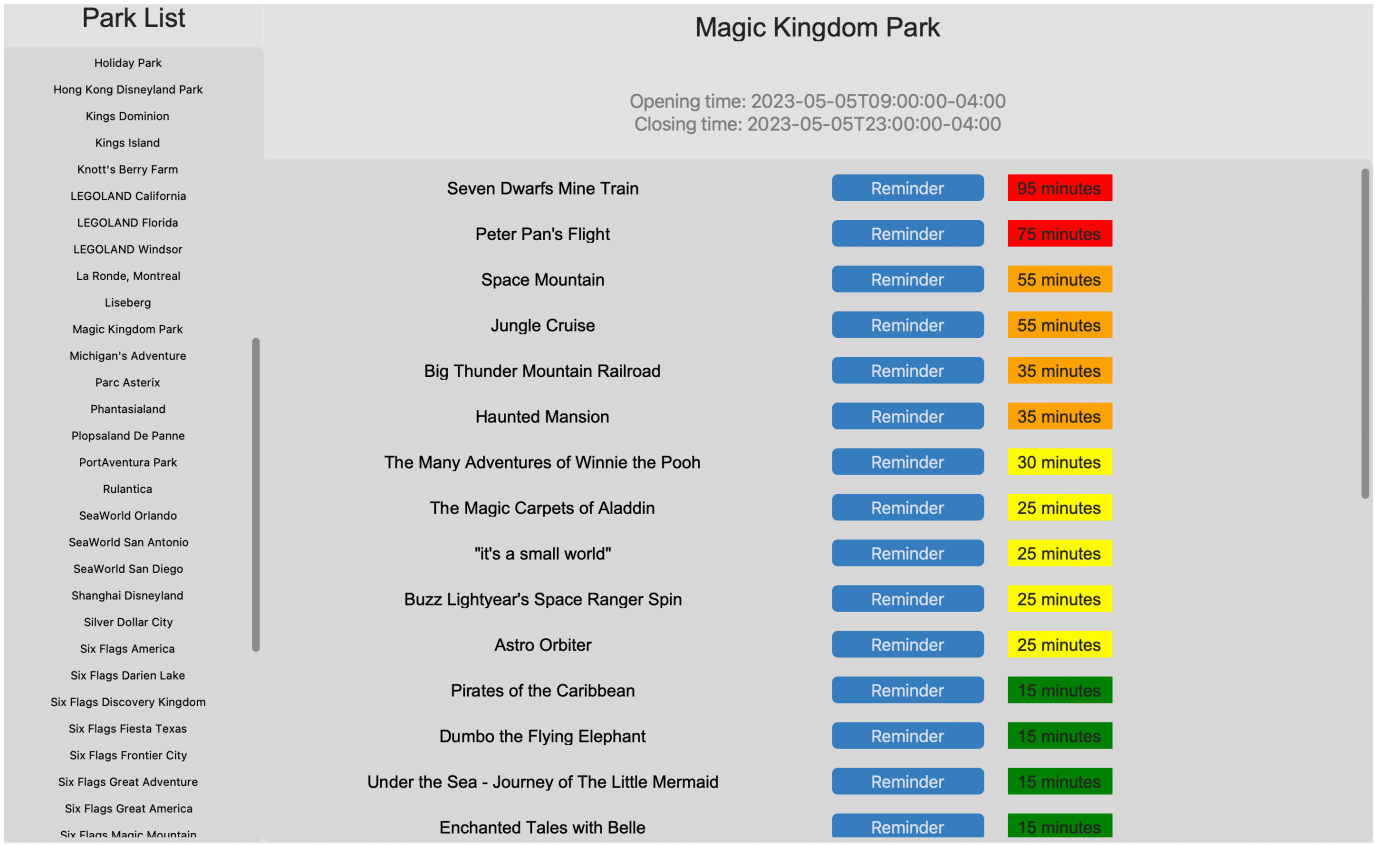
1. 本次实验的第一个难点在于提醒的任务需要多任务并行来完成，即我们希望提醒任务不影响主任务的进行，所以我使用了`MultiProcessing`库，但在使用这个库的时候出现了一个问题，即，我在创建子任务的时候会重新创建一个窗口，最后我们在main函数中加入了`if __name__ == "__main__":`判断语句解决了这个问题
2. 第二个难点在于本次实验使用了`customtkinter`来美化程序界面，并且我们使用了其中例如`TopLevelWindow`的高级功能，在使用这一个库中，没有中文文献参考，读了很多英文文档参考实现了这个界面

实验总结

1. 在GUI设计中，我们应该注意好使用`class`，尽量不把GUI的逻辑写在一个类中，这样可以增加程序的易读性和可维护性
2. GUI的设计中感觉`tkinter`相比于其他设计工具，设计的效率还是不够高，界面也没有想象中的美观
3. 但是本次实验收获了较多，本次实验明白了Python中GUI的主要设计思路，并且学习了如何解析`json`数据
4. 提高了读取英文文档的能力和找取需要的API的能力

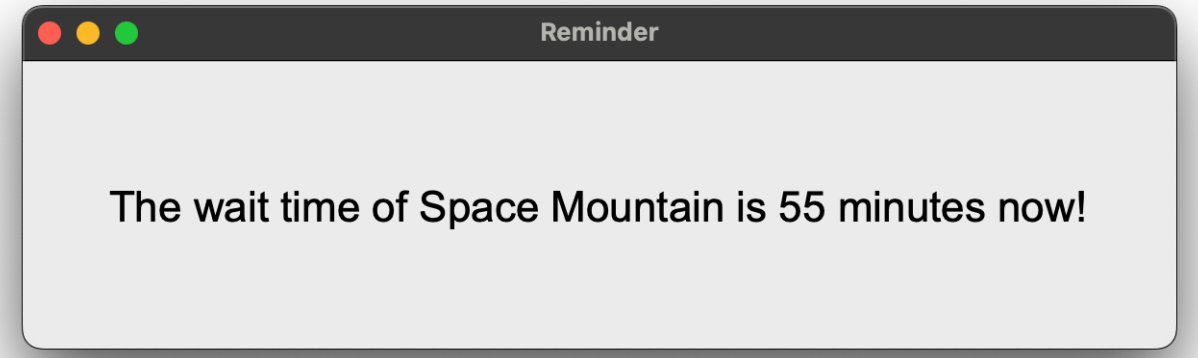
界面布局

1. 本次实验我们的界面布局是这样子的，我们程序左边显示公园列表，右边为主要界面，显示选定的公园各个项目所需要的排队时间，并且颜色会根据排队时长的不同而改变，使得更加醒目



2. 我们可以设置reminder，这会给我们带到第二个界面

3. 这个界面中我们可以选择期待的排队时长，在队伍排队时长低于所需后置顶提醒我们



4. 我制作了一个demo向你展示GUI设计，你可以在那里了解更多