راه‌اندازی:  
۱. ترمینال vscode را باز کنید  
۲. با دستور زیر در ترمینال کتابخانه‌های موردنیاز را نصب کنید

pip install geopy timezonefinder pytz requests

۳. برنامه را با دستور زیر اجرا کنید

python weather.py

توضیحات کلی:

1. \*\*ایمپورت کردن کتابخانه‌ها\*\*

from tkinter import \*

import tkinter as tk

from geopy.geocoders import Nominatim

from tkinter import ttk, messagebox

from timezonefinder import TimezoneFinder

from datetime import datetime

import requests

import pytz

- \*\*`tkinter`\*\*: برای ایجاد رابط کاربری گرافیکی (GUI) استفاده می‌شود. این کتابخانه ابزارهای مختلفی را برای طراحی پنجره‌ها، دکمه‌ها، متن‌ها و غیره فراهم می‌کند.

- \*\*`geopy`\*\*: برای تبدیل نام شهر به مختصات جغرافیایی (طول و عرض جغرافیایی) استفاده می‌شود.

- \*\*`timezonefinder`\*\*: برای پیدا کردن منطقه زمانی بر اساس مختصات جغرافیایی به کار می‌رود.

- \*\*`datetime`\*\*: برای کار با تاریخ و زمان.

- \*\*`requests`\*\*: برای ارسال درخواست به API و دریافت پاسخ.

- \*\*`pytz`\*\*: برای مدیریت مناطق زمانی.

2. \*\*تنظیمات اولیه‌ی پنجره\*\*

```python

root = Tk()

root.title("Weather App!")

root.geometry("900x500+300+200")

root.resizable(False,False)

```

- \*\*`Tk()`\*\*: ایجاد پنجره‌ی اصلی برنامه.

- \*\*`title`\*\*: عنوان پنجره.

- \*\*`geometry`\*\*: تنظیم اندازه‌ی پنجره و موقعیت آن روی صفحه.

- \*\*`resizable`\*\*: جلوگیری از تغییر اندازه‌ی پنجره.

3. \*\*تابع درخواست آب و هوا\*\*

def req\_weather(city\_name):

api\_key = "b9bd320203130831abb3d2b8583d3e79"

base\_url = "http://api.openweathermap.org/data/2.5/weather?"

complete\_url = base\_url + "q=" + city\_name + "&appid=" + api\_key + "&units=metric"

response = requests.get(complete\_url)

return response.json()

- \*\*`req\_weather`\*\*: این تابع یک درخواست به API می‌فرستد تا اطلاعات آب و هوا را بر اساس نام شهر دریافت کند. از کلید API و نام شهر برای ساختن URL استفاده می‌شود و در نهایت پاسخ به فرمت JSON برمی‌گردد.

4. \*\*تابع اصلی برای دریافت و نمایش اطلاعات آب و هوا\*\*

def getWeather():

try:

city = textfield.get()

geolocator = Nominatim(user\_agent="geoapiExercises")

location = geolocator.geocode(city)

obj = TimezoneFinder()

result = obj.timezone\_at(lng=location.longitude, lat=location.latitude)

home = pytz.timezone(result)

local\_time = datetime.now(home)

current\_time = local\_time.strftime("%I:%M %p")

clock.config(text=current\_time)

name.config(text="CURRENT WEATHER")

json\_data = req\_weather(city\_name=city)

condition = json\_data['weather'][0]['main']

description = json\_data['weather'][0]['description']

temp = json\_data['main']['temp']

pressure = json\_data['main']['pressure']

humidity = json\_data['main']['humidity']

wind = json\_data['wind']['speed']

t.config(text=(temp,"°"))

c.config(text=(condition, "|","FEELS","LIKE",temp,"°"))

w.config(text=wind)

h.config(text=humidity)

d.config(text=description)

p.config(text=pressure)

except Exception as e:

messagebox.showerror("Weather App","Invalid Entry!")

- \*\*`getWeather`\*\*: این تابع اطلاعات وارد شده توسط کاربر (نام شهر) را می‌گیرد و از کتابخانه‌های `geopy` و `timezonefinder` برای پیدا کردن مختصات و منطقه زمانی استفاده می‌کند. سپس زمان محلی را نمایش می‌دهد. بعد از آن از تابع `req\_weather` برای دریافت اطلاعات آب و هوا استفاده کرده و این اطلاعات را در رابط کاربری نمایش می‌دهد. اگر خطایی رخ دهد، یک پیام خطا نشان داده می‌شود.

5. \*\*ایجاد المان‌های گرافیکی\*\*

- \*\*فیلد جستجو\*\*، \*\*آیکون جستجو\*\*، \*\*نمایشگر لوگو\*\*، \*\*نمایشگر ساعت و نام شهر\*\*، \*\*و نمایشگر مقادیر آب و هوا\*\* همه در بخش‌های مختلف کد قرار دارند تا اطلاعات آب و هوا به شکل گرافیکی و خوانا نمایش داده شوند.

این بخش‌ها به کمک `Label` ها و `Entry` ها ایجاد می‌شوند و برای هر کدام یک مکان خاص روی پنجره تعریف شده است.

6. \*\*اجرای برنامه\*\*

root.mainloop()

- \*\*`mainloop`\*\*: این خط برنامه را در حالت اجرایی نگه می‌دارد تا کاربر بتواند با رابط کاربری تعامل داشته باشد. تا وقتی که کاربر پنجره را نبندد، برنامه به کار خود ادامه می‌دهد.

----------------------------------------

ریز به ریز tkinter

1. \*\*ایجاد و تنظیم پنجره اصلی\*\*

root = Tk()

root.title("Weather App!")

root.geometry("900x500+300+200")

root.resizable(False, False)

- `Tk()`: ایجاد پنجره اصلی برنامه.

- `title()`: تعیین عنوان پنجره.

- `geometry()`: تنظیم ابعاد و موقعیت پنجره (عرض x ارتفاع + موقعیت افقی + موقعیت عمودی).

- `resizable()`: مشخص می‌کند که آیا پنجره قابل تغییر اندازه است یا نه (در اینجا، تغییر اندازه غیرفعال است).

3. \*\*اجزای رابط کاربری\*\*

جعبه جستجو و دکمه جستجو

Search\_image = PhotoImage(file="search.png")

myimage = Label(image=Search\_image)

myimage.place(x=20, y=20)

- `PhotoImage()`: بارگذاری تصویر از فایل.

- `Label()`: ایجاد یک برچسب (Label) برای نمایش تصویر.

- `place()`: قرار دادن عنصر در موقعیت مشخص شده با مختصات `(x, y)`.

textfield = tk.Entry(root, justify="center", width=17, font=("poppins", 25, "bold"), bg="#404040", border=0, fg="white")

textfield.place(x=50, y=40)

textfield.focus()

- `Entry()`: ایجاد یک فیلد متنی برای ورودی کاربر.

- `justify`: تنظیم تراز متن در فیلد متنی (در اینجا، مرکز).

- `width`: عرض فیلد متنی.

- `font`: تنظیم فونت فیلد متنی.

- `bg`: رنگ پس‌زمینه.

- `border`: تنظیم حاشیه (در اینجا، بدون حاشیه).

- `fg`: رنگ متن.

- `place()`: تعیین موقعیت عنصر.

- `focus()`: تمرکز روی فیلد متنی هنگام بارگذاری برنامه.

Search\_icon = PhotoImage(file="search\_icon.png")

myimage\_icon = Button(image=Search\_icon, borderwidth=0, cursor="hand2", background="#404040", command=getWeather)

myimage\_icon.place(x=400, y=34)

- `Button()`: ایجاد یک دکمه با تصویر.

- `borderwidth`: عرض حاشیه دکمه (در اینجا، بدون حاشیه).

- `cursor`: تعیین نوع نشانگر موس هنگام حرکت روی دکمه.

- `background`: رنگ پس‌زمینه دکمه.

- `command`: تابعی که هنگام کلیک روی دکمه فراخوانی می‌شود.

- `place()`: تعیین موقعیت دکمه.

لوگو و جعبه پایین

Logo\_image = PhotoImage(file="logo.png")

logo = Label(image=Logo\_image)

logo.place(x=150, y=100)

- `Label()`: ایجاد یک برچسب برای نمایش تصویر لوگو.

- `place()`: تعیین موقعیت تصویر لوگو.

Frame\_image = PhotoImage(file="box.png")

frame\_myimage = Label(image=Frame\_image)

frame\_myimage.pack(padx=5, pady=5, side=BOTTOM)

- `pack()`: یک متد دیگر برای چینش عناصر. در اینجا، `pack()` با `padx` و `pady` برای تنظیم فاصله حاشیه‌ای (پدینگ) و `side` برای تعیین موقعیت عنصر در پایین پنجره.

نمایش زمان و اطلاعات آب و هوا

name = Label(root, font=("arial", 15, "bold"))

name.place(x=30, y=100)

- `Label()`: ایجاد برچسب برای نمایش عنوان آب و هوا.

- `place()`: تعیین موقعیت برچسب.

clock = Label(root, font=("Helvetica", 20))

clock.place(x=30, y=130)

- `Label()`: ایجاد برچسب برای نمایش زمان محلی.

- `place()`: تعیین موقعیت برچسب.

برچسب‌ها برای اطلاعات آب و هوا

label1 = Label(root, text="WIND", font=("Helvetica", 15, "bold"), fg="white", bg="#1ab5ef")

label1.place(x=120, y=400)

- `Label()`: ایجاد برچسب برای نمایش عنوان "WIND".

- `text`: متن برچسب.

- `font`: تنظیم فونت.

- `fg`: رنگ متن.

- `bg`: رنگ پس‌زمینه.

- `place()`: تعیین موقعیت برچسب.

t = Label(font=("arial", 70, "bold"), fg="#ee666d")

t.place(x=400, y=150)

- `Label()`: ایجاد برچسب برای نمایش دما.

- `place()`: تعیین موقعیت برچسب.

w = Label(text="....", font=("arial", 20, "bold"), bg="#1ab5ef")

w.place(x=120, y=430)

h = Label(text="....", font=("arial", 20, "bold"), bg="#1ab5ef")

h.place(x=280, y=430)

d = Label(text="....", font=("arial", 20, "bold"), bg="#1ab5ef")

d.place(x=450, y=430)

p = Label(text="....", font=("arial", 20, "bold"), bg="#1ab5ef")

p.place(x=670, y=430)

- `Label()`: ایجاد برچسب‌های برای نمایش اطلاعات آب و هوا (باد، رطوبت، توضیحات، فشار).

- `place()`: تعیین موقعیت برچسب‌ها.

### توضیحات اضافی

- \*\*`config()`\*\*: برای تغییر ویژگی‌های موجود در عنصر استفاده می‌شود، مانند تغییر متن برچسب با `config(text="New Text")`.

- \*\*`place()`\*\*: موقعیت‌یابی دقیق عناصر در پنجره با مختصات `(x, y)`.

- \*\*`pack()`\*\*: چینش عمودی یا افقی عناصر و تنظیم فاصله‌های حاشیه‌ای.

-------------------------

ریز به ریز requests

تابع `req\_weather`

def req\_weather(city\_name):

api\_key = "b9bd320203130831abb3d2b8583d3e79"

base\_url = "http://api.openweathermap.org/data/2.5/weather?"

complete\_url = base\_url + "q=" + city\_name + "&appid=" + api\_key + "&units=metric"

response = requests.get(complete\_url)

return response.json()

توضیحات جزئیات

1. \*\*تعریف تابع\*\*

def req\_weather(city\_name):

- این تابع به نام `req\_weather` تعریف شده است و یک پارامتر ورودی به نام `city\_name` دارد که نام شهری است که می‌خواهید اطلاعات آب و هوا را برای آن دریافت کنید.

2. \*\*کلید API\*\*

api\_key = "b9bd320203130831abb3d2b8583d3e79"

- `api\_key`: کلید API منحصر به فرد است که از وب‌سایت OpenWeatherMap دریافت کرده‌اید. این کلید برای احراز هویت درخواست‌ها به API استفاده می‌شود.

3. \*\*آدرس پایه API\*\*

base\_url = "http://api.openweathermap.org/data/2.5/weather?"

- `base\_url`: این متغیر حاوی آدرس پایه‌ای است که برای درخواست اطلاعات آب و هوا به API OpenWeatherMap استفاده می‌شود. این URL پایه است و باید به آن پارامترهای اضافی اضافه کنید.

4. \*\*ساخت URL کامل\*\*

complete\_url = base\_url + "q=" + city\_name + "&appid=" + api\_key + "&units=metric"

- `complete\_url`: این متغیر URL کامل برای درخواست به API را ایجاد می‌کند. پارامترهای مختلف به این URL اضافه می‌شوند:

- `q=`: نام شهر که به عنوان پارامتر ورودی در URL گنجانده می‌شود.

- `appid=`: کلید API برای احراز هویت.

- `units=metric`: واحد دما را به مقیاس سلسیوس تنظیم می‌کند (می‌توانید از `units=imperial` برای فارنهایت استفاده کنید).

5. \*\*ارسال درخواست GET\*\*

response = requests.get(complete\_url)

- `requests.get()`: این تابع از کتابخانه `requests` برای ارسال یک درخواست GET به URL کامل استفاده می‌شود. این درخواست به سرور OpenWeatherMap ارسال شده و پاسخ آن دریافت می‌شود.

6. \*\*پردازش و بازگرداندن داده‌ها\*\*

return response.json()

- `response.json()`: پاسخ دریافتی از API به فرمت JSON تبدیل می‌شود. این متد داده‌های JSON را به صورت دیکشنری پایتون باز می‌گرداند.

### نحوه استفاده از تابع

در تابع `getWeather()` از `req\_weather()` استفاده می‌شود تا داده‌های آب و هوا برای شهری که کاربر وارد کرده است، دریافت شود و سپس این داده‌ها در رابط کاربری نمایش داده می‌شود.

json\_data = req\_weather(city\_name=city)

- `json\_data`: این متغیر حاوی داده‌های JSON دریافت شده از API است. این داده‌ها شامل اطلاعات مختلفی مانند وضعیت آب و هوا، دما، فشار، رطوبت و سرعت باد است.

### مثال نمونه برای آزمایش

برای تست این تابع، می‌توانید نام یک شهر را به آن بدهید و بررسی کنید که آیا داده‌های صحیح از API دریافت می‌شود یا نه:

data = req\_weather("Tehran")

print(data)

----------------------------------------------------

ریز به ریز کل کتابخانه‌های استفاده شده

### 1. \*\*`tkinter`\*\*

کتابخانه `tkinter` برای ساخت رابط کاربری گرافیکی (GUI) استفاده شده است. این کتابخانه به صورت پیش‌فرض در پایتون وجود دارد و ابزارهایی برای ایجاد پنجره‌ها، دکمه‌ها، برچسب‌ها و دیگر اجزای رابط کاربری ارائه می‌دهد.

- \*\*تعریف و پیکربندی پنجره اصلی\*\*

- `Tk()`: برای ایجاد پنجره اصلی برنامه.

- `title()`, `geometry()`, `resizable()`: برای تنظیم عنوان، ابعاد و قابلیت تغییر اندازه پنجره.

- \*\*ایجاد و تنظیم اجزای رابط کاربری\*\*

- `Label()`, `Entry()`, `Button()`: برای ایجاد برچسب‌ها، فیلدهای متنی و دکمه‌ها.

- `place()`, `pack()`: برای تعیین موقعیت و چینش عناصر در پنجره.

### 2. \*\*`geopy`\*\*

کتابخانه `geopy` برای جغرافیای جغرافیایی و جستجوهای مکانی استفاده می‌شود. در برنامه شما، از `Nominatim` برای یافتن موقعیت جغرافیایی یک شهر استفاده شده است.

- \*\*`Nominatim`\*\*:

- `geocode()`: این متد برای تبدیل نام یک مکان (مانند نام شهر) به مختصات جغرافیایی (عرض جغرافیایی و طول جغرافیایی) استفاده می‌شود.

### 3. \*\*`timezonefinder`\*\*

کتابخانه `timezonefinder` برای تعیین منطقه زمانی بر اساس مختصات جغرافیایی استفاده می‌شود. این کتابخانه به شما کمک می‌کند تا منطقه زمانی مناسب را برای یک مکان خاص پیدا کنید.

- \*\*`TimezoneFinder`\*\*:

- `timezone\_at()`: این متد برای دریافت منطقه زمانی بر اساس طول و عرض جغرافیایی استفاده می‌شود.

### 4. \*\*`datetime`\*\*

کتابخانه `datetime` برای کار با تاریخ و زمان استفاده می‌شود. در برنامه شما، برای دریافت زمان محلی و فرمت کردن آن به فرمت مناسب استفاده شده است.

- \*\*`datetime.now()`\*\*:

- برای دریافت زمان جاری محلی.

- \*\*`strftime()`\*\*:

- برای فرمت کردن زمان به صورت رشته‌ای (مانند "02:30 PM").

### 5. \*\*`requests`\*\*

کتابخانه `requests` برای ارسال درخواست‌های HTTP به API و دریافت پاسخ‌ها استفاده می‌شود. در برنامه شما، از این کتابخانه برای ارسال درخواست به API آب و هوا و دریافت داده‌ها استفاده شده است.

- \*\*`requests.get()`\*\*:

- برای ارسال درخواست GET به URL مشخص شده و دریافت پاسخ.

### 6. \*\*`pytz`\*\*

کتابخانه `pytz` برای کار با مناطق زمانی و تبدیل زمان بین مناطق زمانی مختلف استفاده می‌شود. این کتابخانه به شما کمک می‌کند تا زمان را بر اساس منطقه زمانی خاص تنظیم کنید.

- \*\*`pytz.timezone()`\*\*:

- برای ایجاد یک شیء منطقه زمانی بر اساس نام منطقه زمانی.

### بررسی جزئیات استفاده از هر کتابخانه

\*\*`tkinter`\*\*

برای ایجاد رابط کاربری گرافیکی و چیدمان اجزا. به طور خاص:

- \*\*`Label`\*\*: برای نمایش متن یا تصویر.

- \*\*`Entry`\*\*: برای دریافت ورودی از کاربر.

- \*\*`Button`\*\*: برای ایجاد دکمه‌ها.

- \*\*`place()`, `pack()`\*\*: برای موقعیت‌یابی و چینش اجزا.

\*\*`geopy`\*\*

برای تبدیل نام شهر به مختصات جغرافیایی:

- \*\*`Nominatim`\*\*: برای جستجوی مکانی.

\*\*`timezonefinder`\*\*

برای پیدا کردن منطقه زمانی بر اساس مختصات:

- \*\*`TimezoneFinder`\*\*: برای دریافت نام منطقه زمانی از مختصات.

\*\*`datetime`\*\*

برای کار با زمان و تاریخ:

- \*\*`datetime.now()`\*\*: برای دریافت زمان فعلی.

- \*\*`strftime()`\*\*: برای فرمت کردن زمان به رشته.

\*\*`requests`\*\*

برای ارسال درخواست‌های HTTP:

- \*\*`requests.get()`\*\*: برای ارسال درخواست GET و دریافت پاسخ.

\*\*`pytz`\*\*

برای تنظیم زمان محلی بر اساس منطقه زمانی:

- \*\*`pytz.timezone()`\*\*: برای ایجاد شیء زمان بر اساس نام منطقه زمانی.

—------------------------

\*\*مراحل تبدیل برنامه پایتون به EXE با cx\_Freeze\*\*

\*\*1. نصب cx\_Freeze\*\*

برای نصب `cx\_Freeze`، از دستور زیر در ترمینال یا خط فرمان استفاده کنید:

pip install cx\_Freeze

\*\*2. ایجاد فایل `setup.py`\*\*

فایل `setup.py` برای تنظیمات و پیکربندی‌های لازم جهت تبدیل برنامه پایتون به فایل EXE استفاده می‌شود. این فایل باید در دایرکتوری اصلی پروژه شما ایجاد شود.

\*\*محتوای فایل `setup.py`:\*\*

from cx\_Freeze import setup, Executable

import sys

# تنظیمات اضافی

build\_exe\_options = {

"packages": ["tkinter", "geopy", "timezonefinder", "requests", "pytz"], # لیست پکیج‌هایی که باید شامل شوند

"include\_files": ["search.png", "search\_icon.png", "logo.png", "box.png"] # فایل‌های اضافی مانند تصاویر

}

# تنظیمات اصلی

setup(

name = "Weather App", # نام برنامه

version = "1.0", # نسخه برنامه

description = "A weather application.", # توضیحات برنامه

options = {"build\_exe": build\_exe\_options}, # تنظیمات اضافی

executables = [Executable("weather\_app.py", base=None)] # فایل پایتون که باید به EXE تبدیل شود

)

\*\*توضیحات:\*\*

- \*\*`build\_exe\_options`\*\*:

- \*\*`packages`\*\*: لیست پکیج‌هایی که برنامه به آن‌ها نیاز دارد. این پکیج‌ها به طور خودکار شناسایی می‌شوند و در EXE گنجانده می‌شوند.

- \*\*`include\_files`\*\*: فایل‌های اضافی مانند تصاویر که باید در کنار فایل EXE قرار گیرند. این فایل‌ها باید در همان دایرکتوری که `setup.py` قرار دارد موجود باشند.

- \*\*`setup()`\*\*:

- \*\*`name`\*\*: نام برنامه‌ای که در EXE نمایش داده می‌شود.

- \*\*`version`\*\*: نسخه برنامه.

- \*\*`description`\*\*: توضیحات مختصر درباره برنامه.

- \*\*`options`\*\*: شامل تنظیمات اضافی مانند پکیج‌ها و فایل‌های اضافی.

- \*\*`executables`\*\*: لیست فایل‌های پایتون که باید به EXE تبدیل شوند. در اینجا فقط یک فایل پایتون (`weather\_app.py`) وجود دارد.

\*\*3. اجرای فرآیند ساخت\*\*

پس از ایجاد فایل `setup.py`, به دایرکتوری که فایل `setup.py` در آن قرار دارد بروید و دستور زیر را در ترمینال یا خط فرمان اجرا کنید:

python setup.py build

\*\*توضیحات:\*\*

- \*\*`python setup.py build`\*\*: این دستور باعث می‌شود `cx\_Freeze` فایل‌های مورد نیاز را جمع‌آوری کرده و یک فایل EXE در دایرکتوری `build` ایجاد کند.