Creating the GFS6p Web API

By: Bas Retsios

Date: 25-November-2021

Goal

Develop a Web API service that provides the 10-day Global Forecast System (GFS) data for a single lat/lon location, instantly, and requiring minimal internet bandwidth.

Introduction

Global Forecast System wikipedia documentation:
https://en.wikipedia.org/wiki/Global_Forecast_System

Global Forecast System official website:

https://www.ncei.noaa.gov/products/weather-climate-models/global-forecast

Global Forecast System data download link: https://www.nco.ncep.noaa.gov/pmb/products/gfs/

Downloading 10-day forecast files (10 raster images for 6 parameters) takes 2 hours. Every 6 hours a new forecast is available.

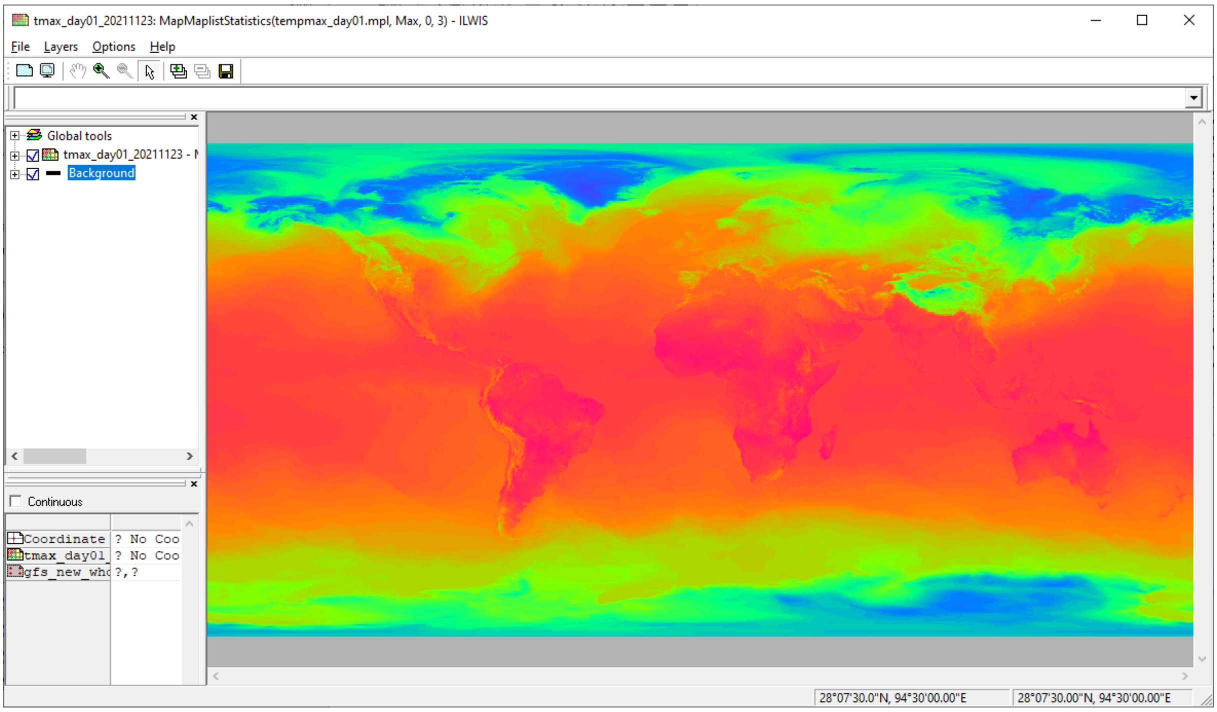
At the ITC, a service is downloading the forecast files (daily, at 6 AM), and aggregates the 6-hourly forecasts to daily. The result is available here: https://filetransfer.itc.nl/pub/mpe/gfs_6p/

All forecast data files are available at this location, from the beginning that the service became operational, in September 2018.

For this Web API service, we are only interested in the most recent data file, which becomes available (after downloading, aggregating and processing) around 8:30 AM every day. This is based on the forecast data that was available the previous day.

To get an impression on what is available, download the most recent data file, unzip it, and use ILWIS to view the content (the data files are stored in the ILWIS format).

Below an example of the tmax (maximum temperature) file produced on 23-November 2021. This is the forecast of day-1 in the 10-day period, so this is the temperature-forecast of 23-November-2021.



tmax_day01_20211123: MapMaplistStatistics(tempmax_day01.mpl, Max, 0, 3) - ILWIS

Coordinate	? No Coo
tmax_day01_	? No Coo
gfs_new_who	?,?

28°07'30.0"N, 94°30'00.00"E

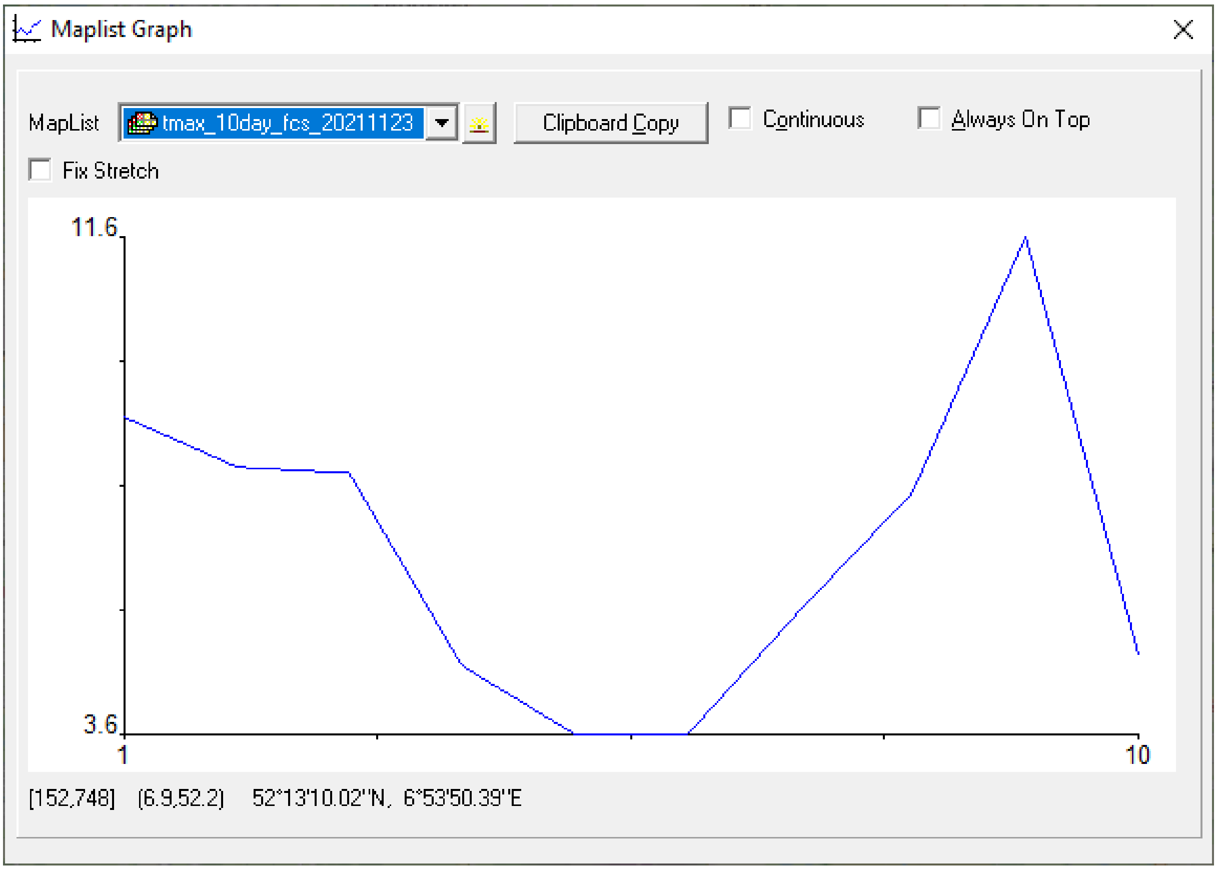
28°07'30.00"N, 94°30'00.00"E

6 parameters are available. Those are the forecasts of:

- tmin: daily minimum temperature
- tmax: daily maximum temperature
- apcp/Prec: Rainfall
- lhtfl/ETa: actual evapotranspiration
- pevpr/ETo: potential evapotranspiration
- rh: Relative Humidity

The goal of the Web API is – given a location expressed in latitude and longitude, to deliver for each parameter the 10 forecast values for the next 10 days.

Example for the tmax forecast of 23-November-2021, at lat=52.219611, lon=6.896532 (Enschede, NL). The maximum temperature from November 23 til December 2 is predicted to be between 3.6 and 11.6 degrees Celsius.



[152,748] (6.9,52.2) 52°13'10.02"N, 6°53'50.39"E

Thus given lat=52.219611 and lon=6.896532, the Web API must return the list with the daily temperature numbers: [8.7, 7.9, 7.8, 4.7, 3.6, 3.6, 5.6, 7.5, 11.6, 4.9], so that a client of this Web API can plot the temperature graphic.

In fact, the Web API must return the daily numbers of all 6 parameters: tmin, tmax, apcp, lhtfl, pevpr, rh. It is then up to the client what to do with them (plot all or a selection).

Daily automation script

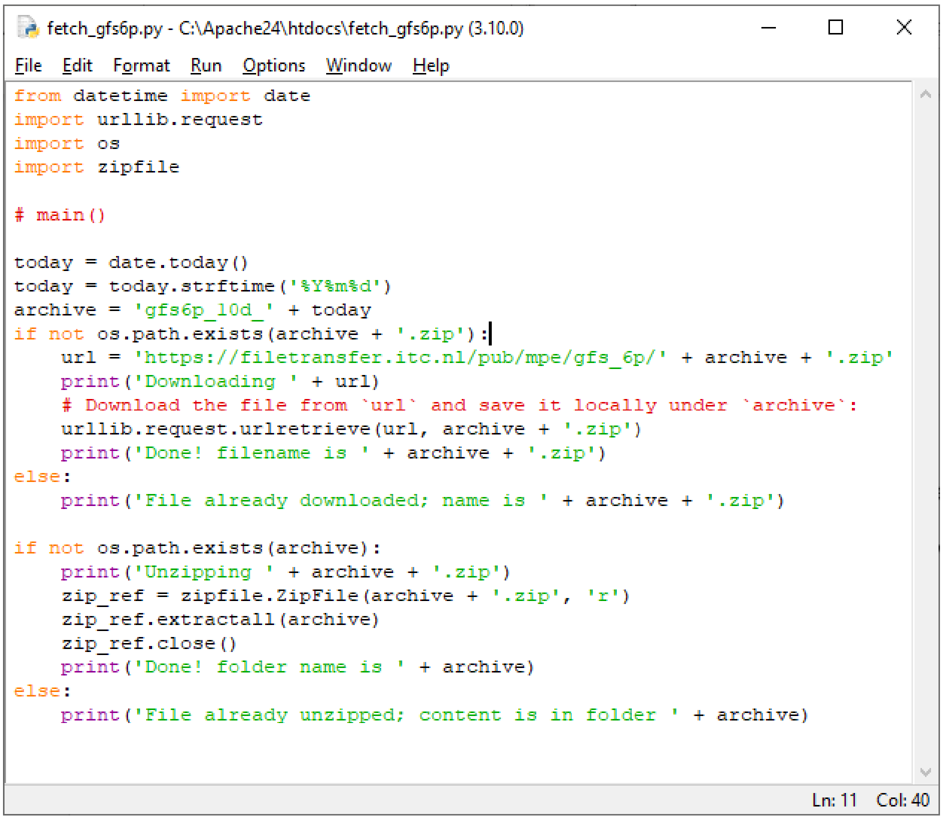
In order to maximize the performance of the Web API, and thus avoid downloading and unzipping a file for every Web API request, a (Python) script will be created that automatically downloads and unzips the latest forecast file from https://filetransfer.itc.nl/pub/mpe/gfs_6p/ . The script must run once a day, in the morning, after the processing script has downloaded from GFS, aggregated the data, and placed the zipfile at this location. The correct time for execution is around 8:30 AM. No further download or preprocessing is needed for requests to the Web API, until the next day.

The following is the python code that will download and unzip the daily zipfile. Start your newly installed IDLE (the Python Interactive Development Environment), select File -> New File, and copy/paste the python code in the new file. Save the file in folder C:\Apache24\htdocs , and name it fetch_gfs6p.py .

```
from datetime import date

import urllib.request

import os
import zipfile


# main()


today = date.today()

today = today.strftime('%Y%m%d')

archive = 'gfs6p_10d_' + today

if not os.path.exists(archive + '.zip'):

    url = 'https://filetransfer.itc.nl/pub/mpe/gfs_6p/' + archive +
'.zip'

    print('Downloading ' + url)

    # Download the file from `url` and save it locally under
`archive`:

    urllib.request.urlretrieve(url, archive + '.zip')

    print('Done! filename is ' + archive + '.zip')

else:

    print('File already downloaded; name is ' + archive + '.zip')


if not os.path.exists(archive):

    print('Unzipping ' + archive + '.zip')

    zip_ref = zipfile.ZipFile(archive + '.zip', 'r')

    zip_ref.extractall(archive)

    zip_ref.close()

    print('Done! folder name is ' + archive)

else:

    print('File already unzipped; content is in folder ' + archive)
```

Place this text in a new python script (named fetch_gfs6p.py) in folder C:\Apache24\htdocs .

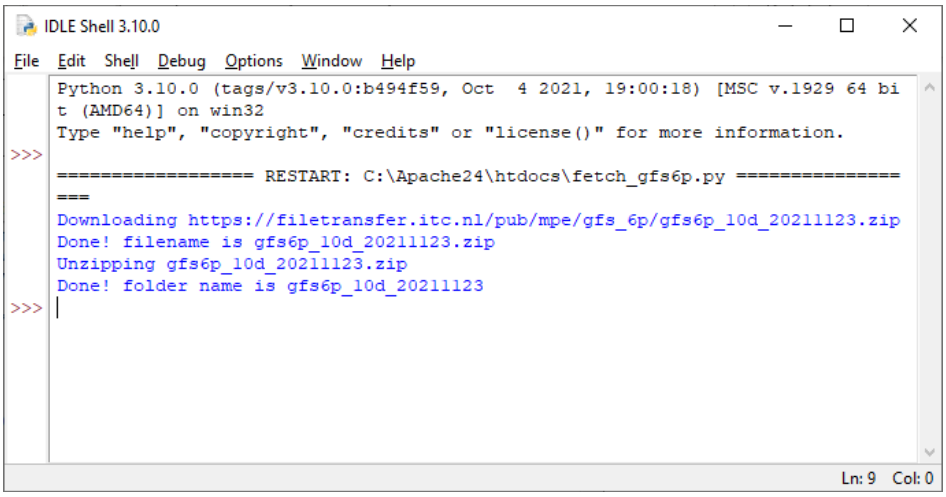
Observe that the file uses the date of today to “guess” the filename that must be downloaded.



fetch_gfs6p.py - C:\Apache24\htdocs\fetch_gfs6p.py (3.10.0)
File Edit Format Run Options Window Help
Ln: 11 Col: 40

```python
from datetime import date
import urllib.request
import os
import zipfile

# main()

today = date.today()
today = today.strftime('%Y%m%d')
archive = 'gfs6p_10d_' + today
if not os.path.exists(archive + '.zip'):
    url = 'https://filetransfer.itc.nl/pub/mpe/gfs_6p/' + archive + '.zip'
    print('Downloading ' + url)
    # Download the file from `url` and save it locally under `archive`:
    urllib.request.urlretrieve(url, archive + '.zip')
    print('Done! filename is ' + archive + '.zip')
else:
    print('File already downloaded; name is ' + archive + '.zip')

if not os.path.exists(archive):
    print('Unzipping ' + archive + '.zip')
    zip_ref = zipfile.ZipFile(archive + '.zip', 'r')
    zip_ref.extractall(archive)
    zip_ref.close()
    print('Done! folder name is ' + archive)
else:
    print('File already unzipped; content is in folder ' + archive)
```

If the time is already 8:30 AM, test the file by running it (Run -> Run Module).

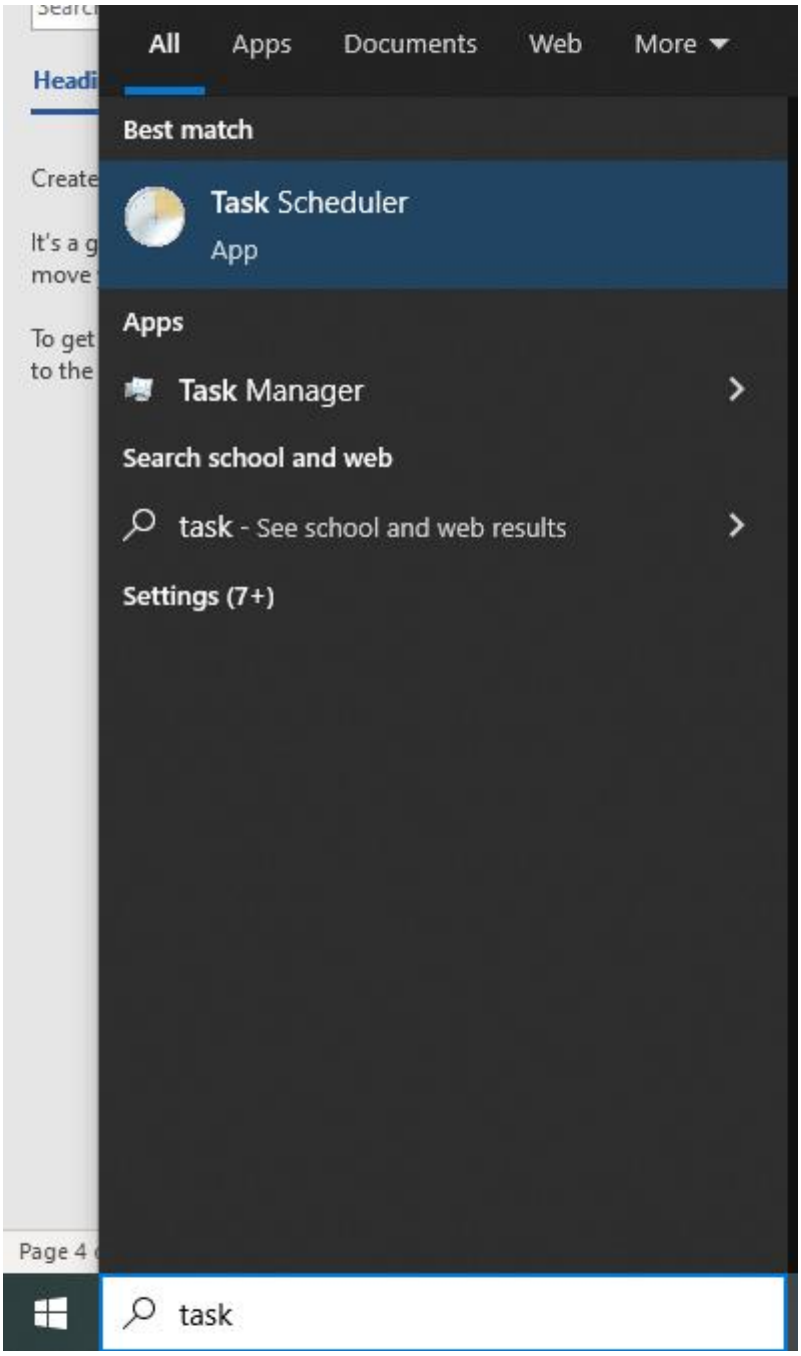


```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bi
t (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Apache24\htdocs\fetch_gfs6p.py ===============
===
Downloading https://filetransfer.itc.nl/pub/mpe/gfs_6p/gfs6p_10d_20211123.zip
Done! filename is gfs6p_10d_20211123.zip
Unzipping gfs6p_10d_20211123.zip
Done! folder name is gfs6p_10d_20211123
>>>
```

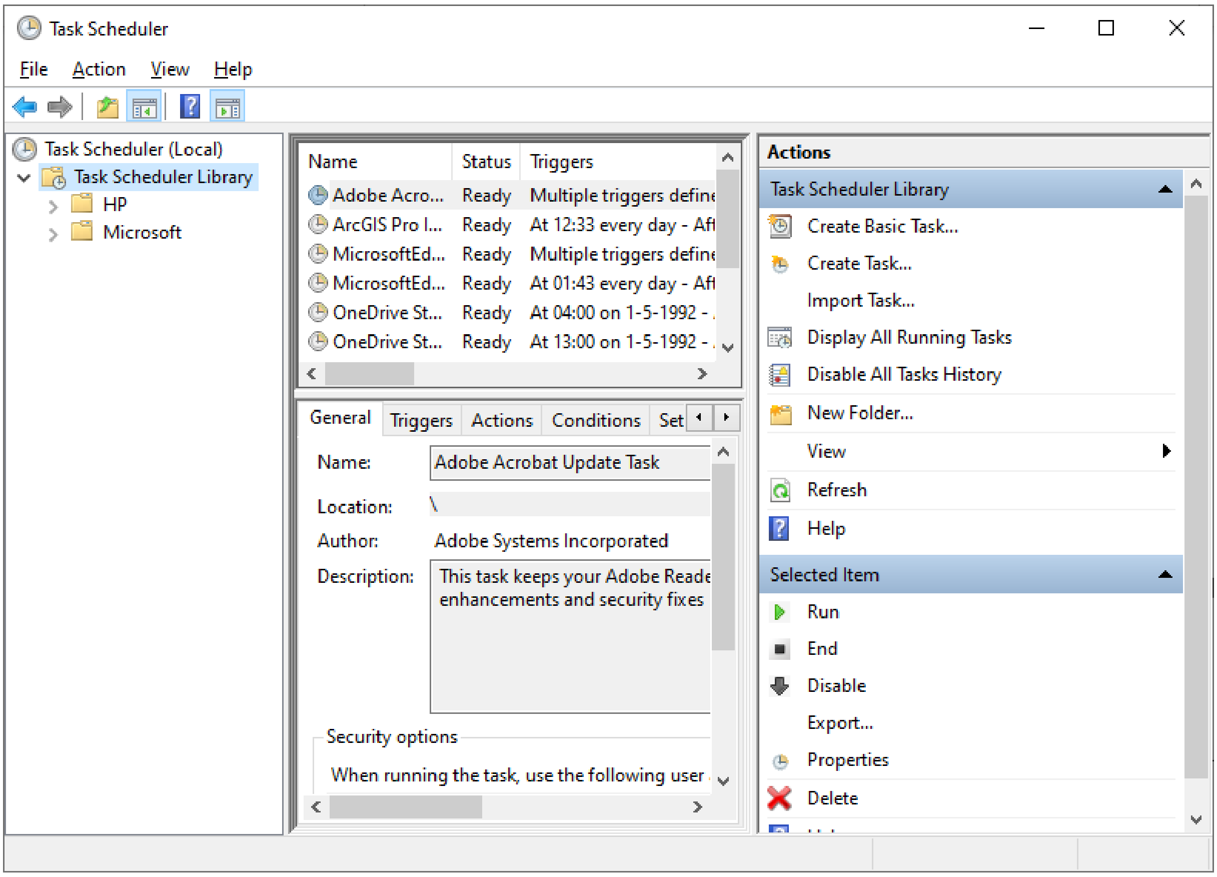
Observe that inside folder C:\Apache24\htdocs a new folder is created, containing the ILWIS raster images (10 images for 6 parameters; in total 60 raster images and some metadata files).

The above script can be scheduled to run automatically every day at the same time, using the Windows Task Scheduler. To open the Task Scheduler, press the Windows button, and type ‘task’.

Task Scheduler should appear as one of the options. Open it.



Click on Task Scheduler Library to observe the existing tasks that have been configured:



Name	Status	Triggers
Adobe Acro...	Ready	Multiple triggers define
ArcGIS Pro I...	Ready	At 12:33 every day - Aft
MicrosoftEd...	Ready	Multiple triggers define
MicrosoftEd...	Ready	At 01:43 every day - Aft
OneDrive St...	Ready	At 04:00 on 1-5-1992 - .
OneDrive St...	Ready	At 13:00 on 1-5-1992 - .

Name: Adobe Acrobat Update Task

Location: \

Author: Adobe Systems Incorporated

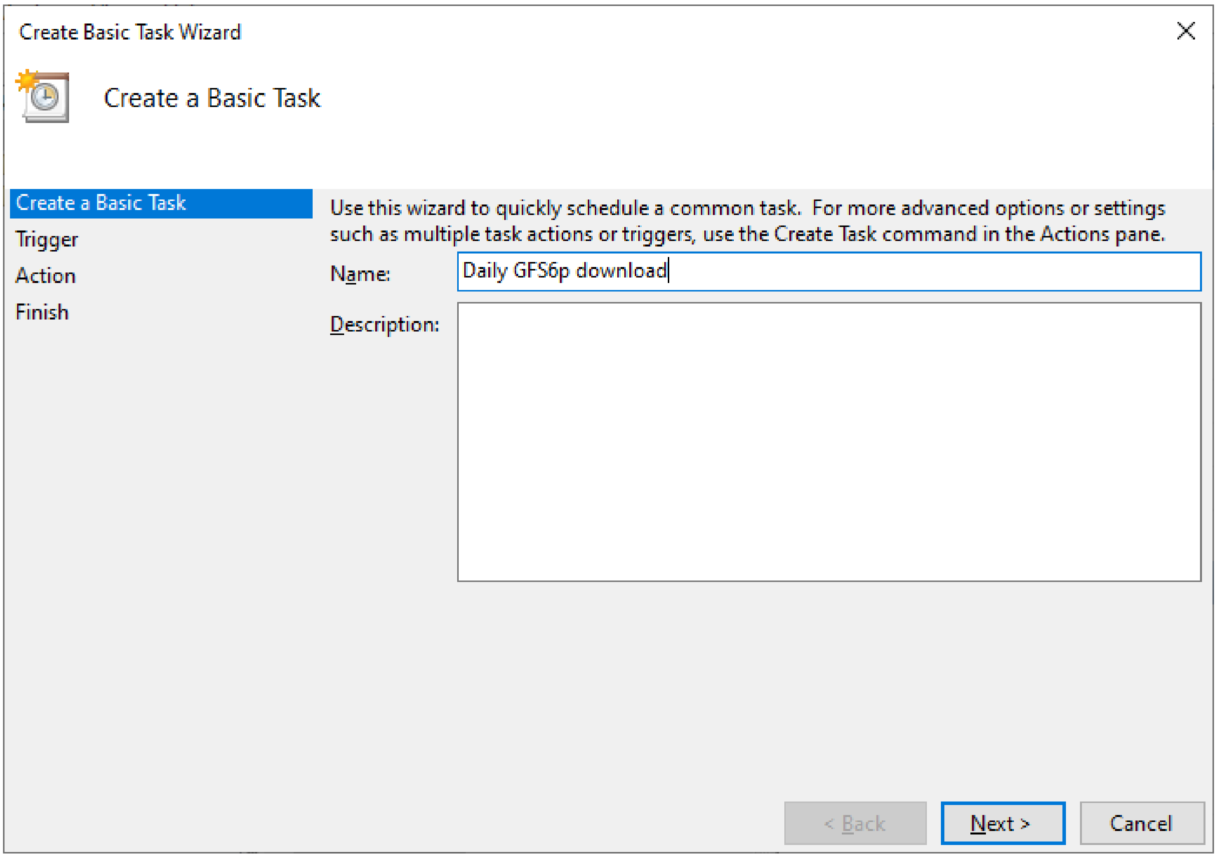
Description: This task keeps your Adobe Reade enhancements and security fixes

Security options

When running the task, use the following user .

At the right side, click “Create Basic Task…”.

Name: give it a name that distinguishes this task.

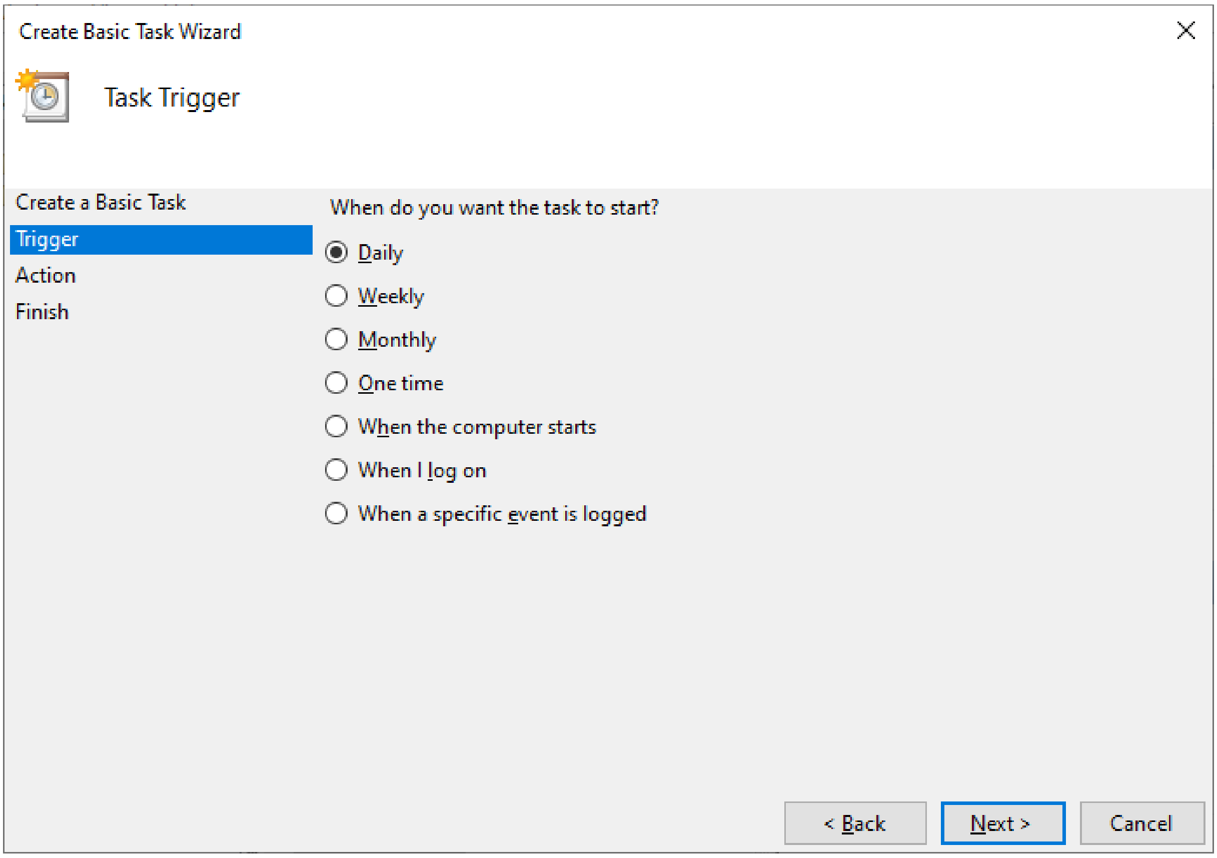


Use this wizard to quickly schedule a common task. For more advanced options or settings such as multiple task actions or triggers, use the Create Task command in the Actions pane.

Name: Daily GFS6p download

Description:

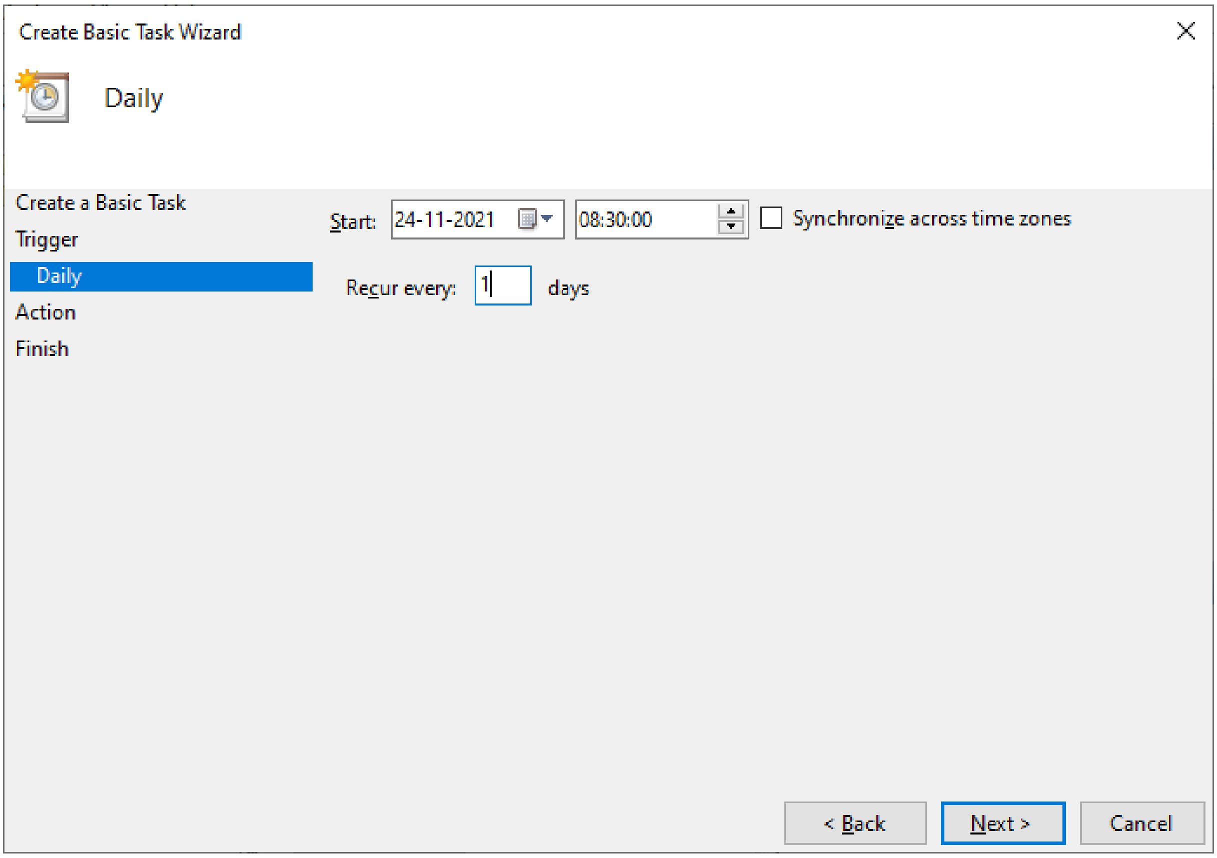
Trigger: daily



When do you want the task to start?

● Daily
○ Weekly
○ Monthly
○ One time
○ When the computer starts
○ When I log on
○ When a specific event is logged

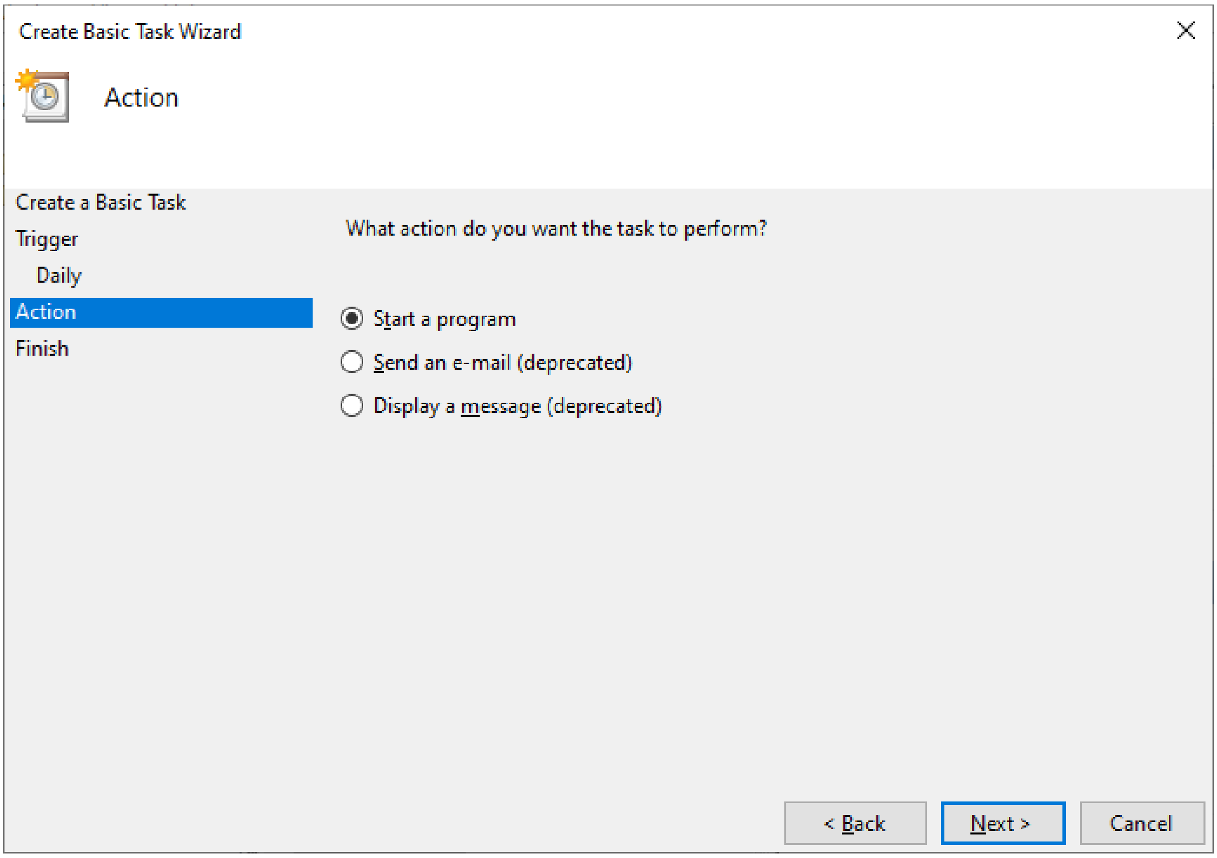
Start: 08:30 every day:



Start: 24-11-2021 08:30:00 □ Synchronize across time zones

Recur every: 1 days

Action: start a program



Create Basic Task Wizard

Action

What action do you want the task to perform?

● Start a program

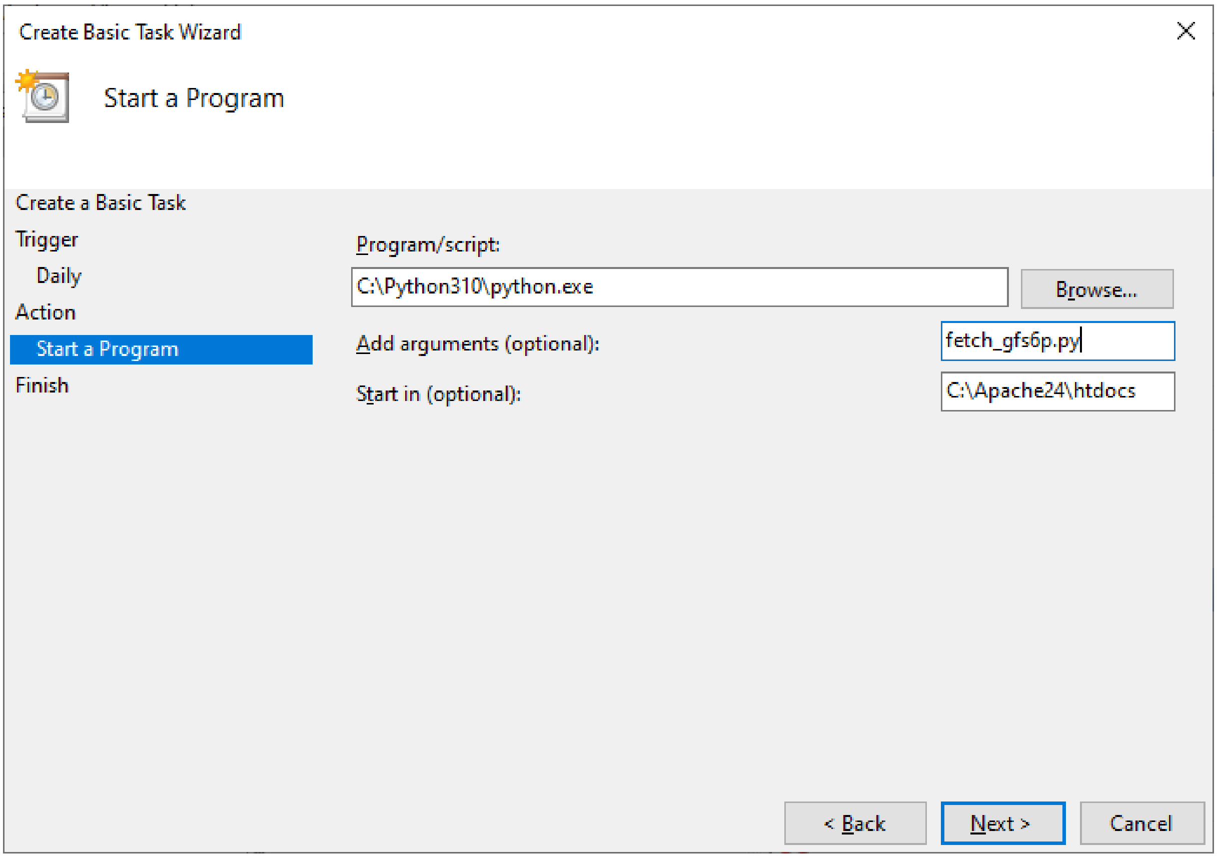
○ Send an e-mail (deprecated)

○ Display a message (deprecated)

Program: python.exe

Arguments: the name of the script to be executed: fetch_gfs6p.py

Start-in (the location of the script and the data): C:\Apache24\htdocs



Create Basic Task Wizard

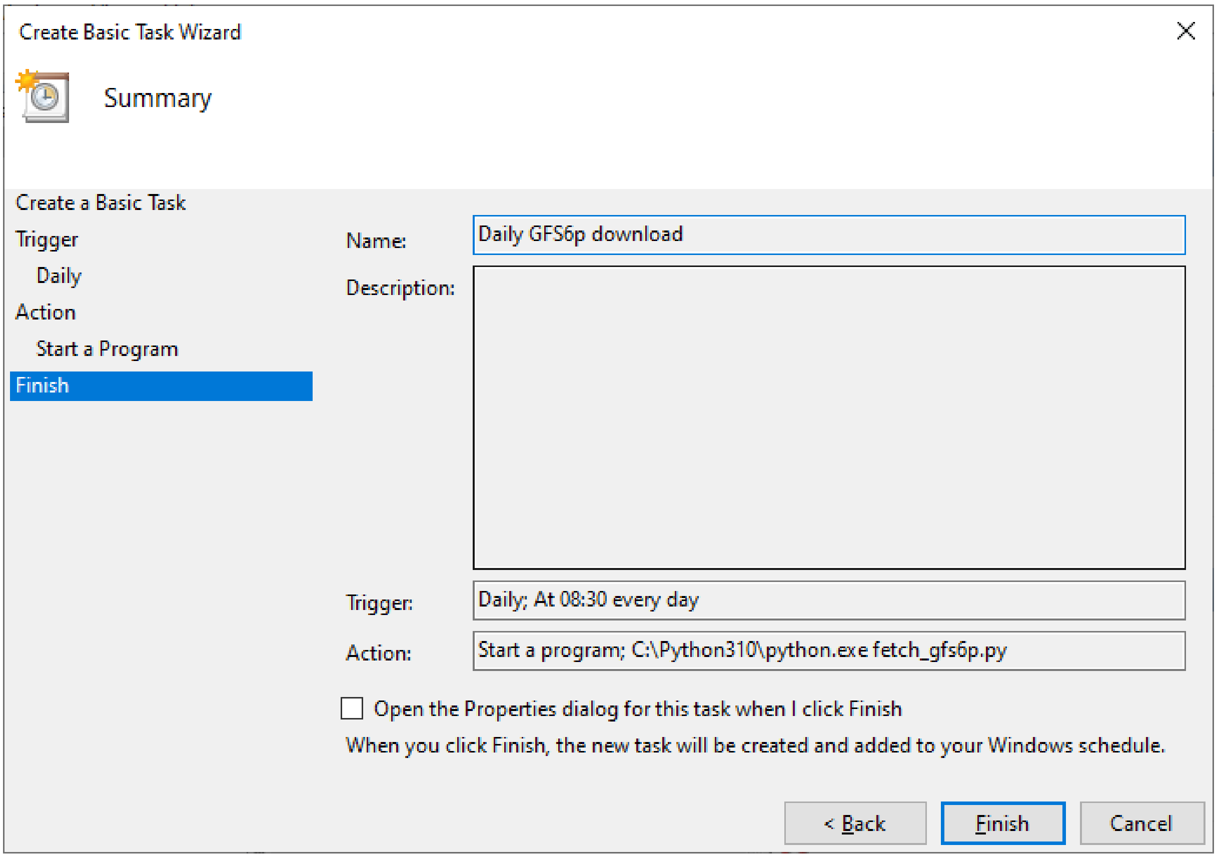
Start a Program

Program/script: C:\Python310\python.exe

Add arguments (optional): fetch_gfs6p.py

Start in (optional): C:\Apache24\htdocs

Click Finish.



Create Basic Task Wizard

Summary

Name: Daily GFS6p download

Description:

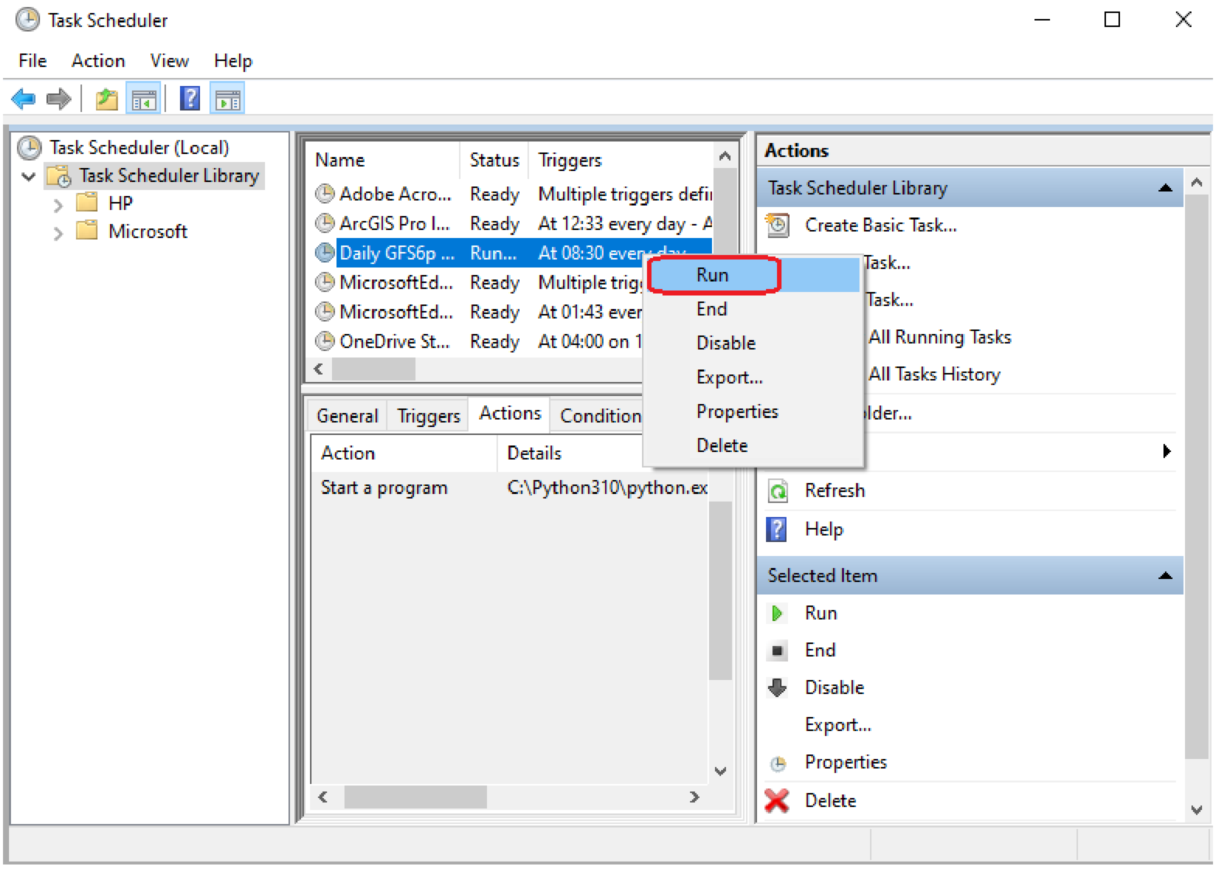
Trigger: Daily; At 08:30 every day

Action: Start a program; C:\Python310\python.exe fetch_gfs6p.py

□ Open the Properties dialog for this task when I click Finish

When you click Finish, the new task will be created and added to your Windows schedule.

The task is added in the list. Right-click on it and select Run to test it:



Name	Status	Triggers
Adobe Acro...	Ready	Multiple triggers defi
ArcGIS Pro I...	Ready	At 12:33 every day - A
Daily GFS6p ...	Run...	At 08:30 every day
MicrosoftEd...	Ready	Multiple trig
MicrosoftEd...	Ready	At 01:43 ever
OneDrive St...	Ready	At 04:00 on 1

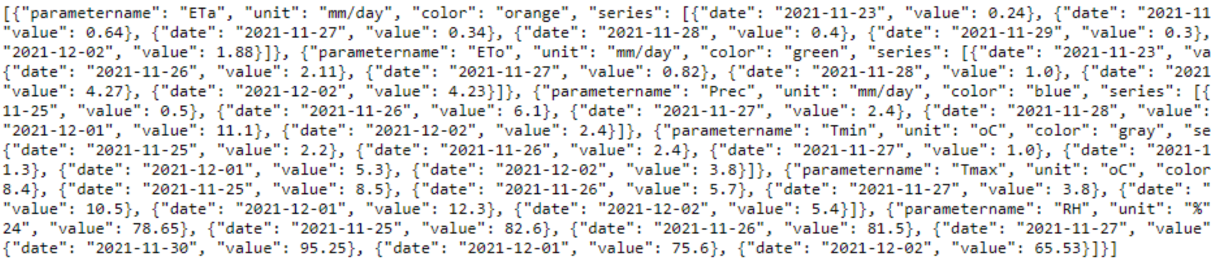
Action	Details
Start a program	C:\Python310\python.ex

This would fetch the gfs6p zipfile (but the script is made to detect whether the file is already downloaded, so it is not downloaded twice).

Web API python script

Now create the main Web API script that – given a latitude and longitude – will return 6 lists (one for each parameter) with 10 numbers each (one for each day forecasted). To make the output of the script more widely usable (to make it e.g. easier to implement a Web-Client), the output must be in JSON format (instead of plain text or comma-delimited).

Here is an example of JSON formatted output (with hierarchical name/value pairs). In python such a structure is called a “dictionary”.



```
[{"parametername": "ETa", "unit": "mm/day", "color": "orange", "series": [{"date": "2021-11-23", "value": 0.24}, {"date": "2021-11
"value": 0.64}, {"date": "2021-11-27", "value": 0.34}, {"date": "2021-11-28", "value": 0.4}, {"date": "2021-11-29", "value": 0.3},
"2021-12-02", "value": 1.88}]}, {"parametername": "ETo", "unit": "mm/day", "color": "green", "series": [{"date": "2021-11-23", "va
{"date": "2021-11-26", "value": 2.11}, {"date": "2021-11-27", "value": 0.82}, {"date": "2021-11-28", "value": 1.0}, {"date": "2021
"value": 4.27}, {"date": "2021-12-02", "value": 4.23}]}, {"parametername": "Prec", "unit": "mm/day", "color": "blue", "series": [{
11-25", "value": 0.5}, {"date": "2021-11-26", "value": 6.1}, {"date": "2021-11-27", "value": 2.4}, {"date": "2021-11-28", "value":
"2021-12-01", "value": 11.1}, {"date": "2021-12-02", "value": 2.4}]}, {"parametername": "Tmin", "unit": "oC", "color": "gray", "se
{"date": "2021-11-25", "value": 2.2}, {"date": "2021-11-26", "value": 2.4}, {"date": "2021-11-27", "value": 1.0}, {"date": "2021-1
1.3}, {"date": "2021-12-01", "value": 5.3}, {"date": "2021-12-02", "value": 3.8}]}, {"parametername": "Tmax", "unit": "oC", "color
8.4}, {"date": "2021-11-25", "value": 8.5}, {"date": "2021-11-26", "value": 5.7}, {"date": "2021-11-27", "value": 3.8}, {"date": "
"value": 10.5}, {"date": "2021-12-01", "value": 12.3}, {"date": "2021-12-02", "value": 5.4}]}, {"parametername": "RH", "unit": "%"
24", "value": 78.65}, {"date": "2021-11-25", "value": 82.6}, {"date": "2021-11-26", "value": 81.5}, {"date": "2021-11-27", "value"
{"date": "2021-11-30", "value": 95.25}, {"date": "2021-12-01", "value": 75.6}, {"date": "2021-12-02", "value": 65.53}]}]
```

Step 1: capture the correct GFS binary data files

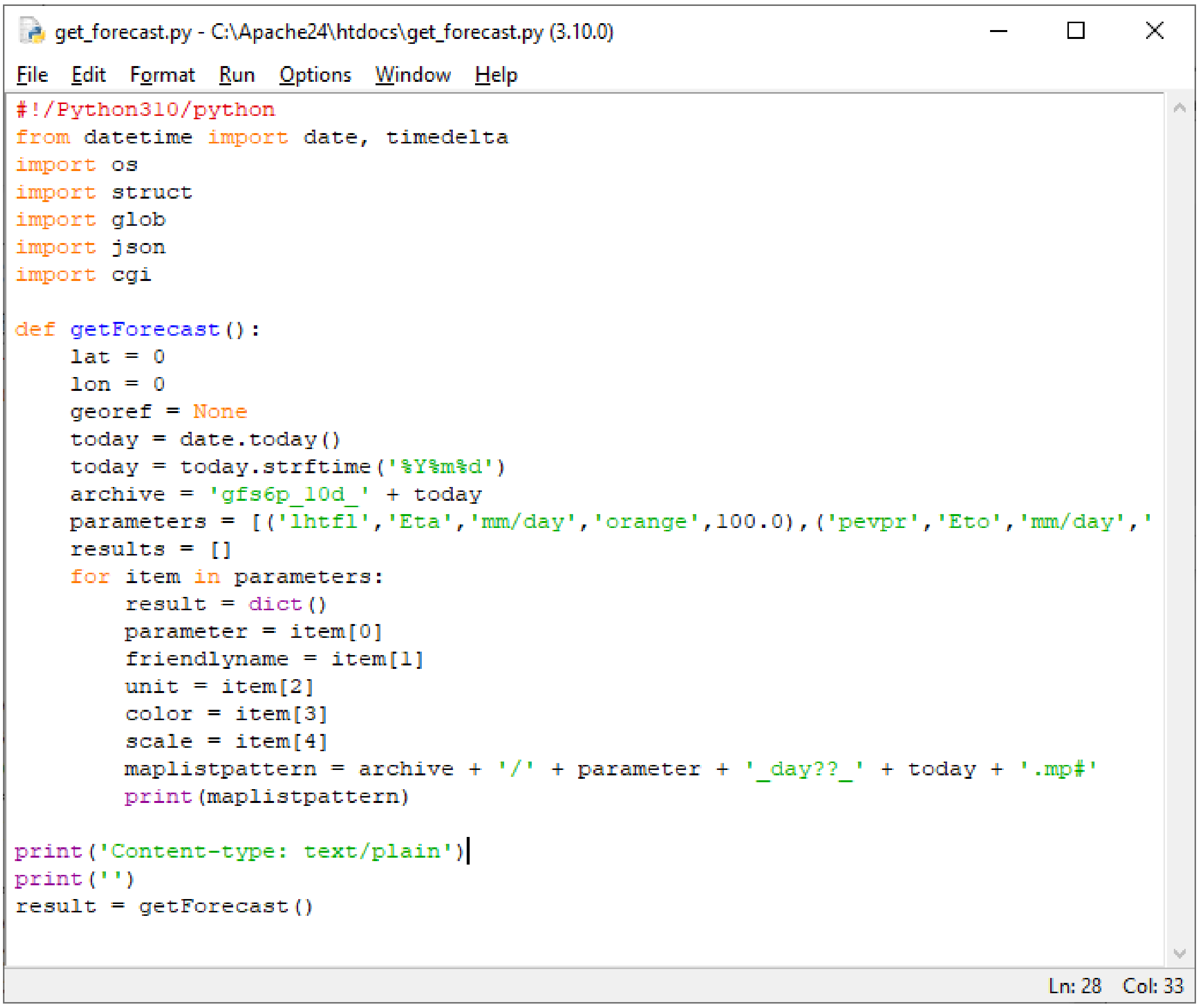
At the same location as the previously created script test.py (C:\Apache24\htdocs), use IDLE to create a new empty python file, and name it get_forecast.py .

Place the following code as the initial content of this new file:

```
#!/Python310/python
from datetime import date, timedelta
import os
import struct
import glob
import json
import cgi

def getForecast():
    lat = 0
    lon = 0
    georef = None
    today = date.today()
    today = today.strftime('%Y%m%d')
    archive = 'gfs6p_10d_' + today
    parameters =
[('lhtfl','ETa','mm/day','orange',100.0),('pevpr','ETo','mm/day','gr
een',100.0),('apcp','Prec','mm/day','blue',10.0),('tmin','Tmin','oC'
,'gray',10.0),('tmax','Tmax','oC','red',10.0),('rh','RH','%','cyan',
100.0)]
    results = []
    for item in parameters:
        result = dict()
        parameter = item[0]
        friendlyname = item[1]
        unit = item[2]
        color = item[3]
        scale = item[4]
        maplistpattern = archive + '/' + parameter + '_day??_' +
today + '.mp#'
        print(maplistpattern)

print('Content-type: text/plain')
print('')
```

```
result = getForecast()
```

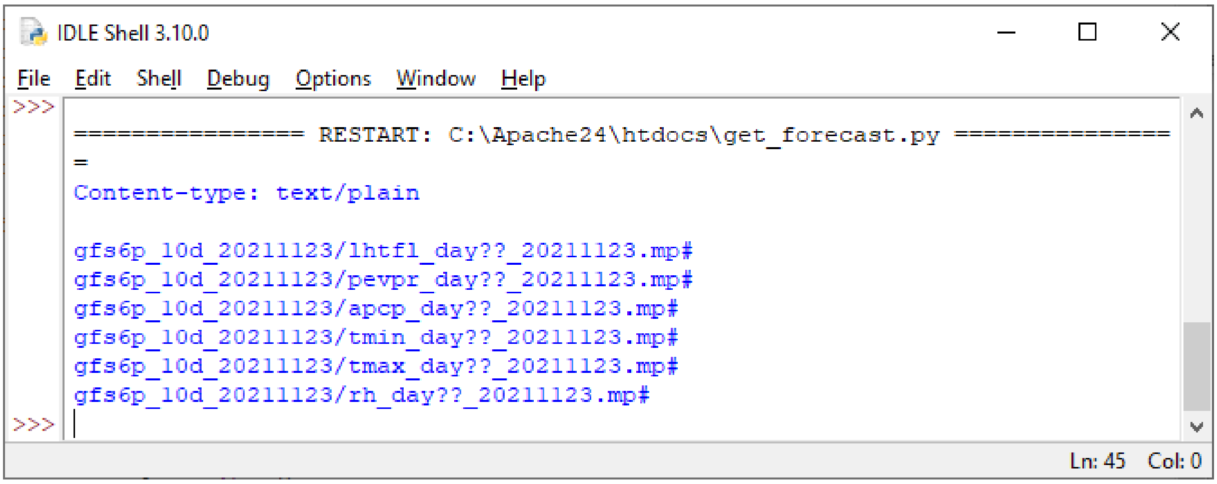
The script will look like this:


get_forecast.py - C:\Apache24\htdocs\get_forecast.py (3.10.0)
File Edit Format Run Options Window Help
Ln: 28 Col: 33

```
#!/Python310/python
from datetime import date, timedelta
import os
import struct
import glob
import json
import cgi

def getForecast():
    lat = 0
    lon = 0
    georef = None
    today = date.today()
    today = today.strftime('%Y%m%d')
    archive = 'gfs6p_10d_' + today
    parameters = [('lhtfl','Eta','mm/day','orange',100.0),('pevpr','Eto','mm/day','
    results = []
    for item in parameters:
        result = dict()
        parameter = item[0]
        friendlyname = item[1]
        unit = item[2]
        color = item[3]
        scale = item[4]
        maplistpattern = archive + '/' + parameter + '_day??_' + today + '.mp#'
        print(maplistpattern)

print('Content-type: text/plain')
print('')
result = getForecast()
```

Run the script (Run -> Run Module).

Confirm the text-output of the script. Those are the file-patterns that would match the 10 files for each parameter. Confirm that those files are available in folder C:\Apache24\htdocs\gfs6p_10d_2021MMDD, replace MMDD with the actual date).



```
>>>
    ================ RESTART: C:\Apache24\htdocs\get_forecast.py ===============
    =
    Content-type: text/plain

    gfs6p_10d_20211123/lhtfl_day??_20211123.mp#
    gfs6p_10d_20211123/pevpr_day??_20211123.mp#
    gfs6p_10d_20211123/apcp_day??_20211123.mp#
    gfs6p_10d_20211123/tmin_day??_20211123.mp#
    gfs6p_10d_20211123/tmax_day??_20211123.mp#
    gfs6p_10d_20211123/rh_day??_20211123.mp#
>>>
```

Step 2: list all filenames

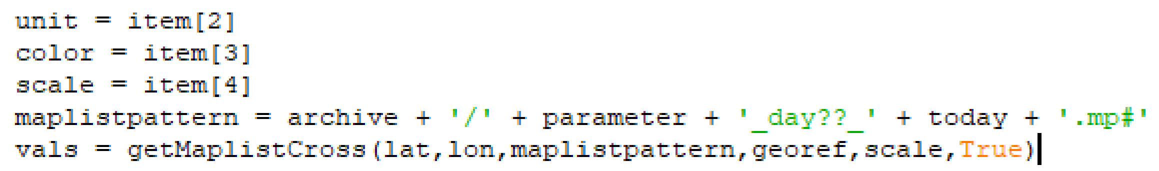
Back in the IDLE editor, replace the line:

```
print(maplistpattern)
```

with the following:

```
vals = getMaplistCross(lat,lon,maplistpattern,georef,scale,True)
```

Result:

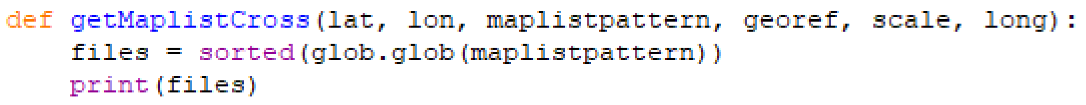


```
unit = item[2]
color = item[3]
scale = item[4]
maplistpattern = archive + '/' + parameter + '_day??_' + today + '.mp#'
vals = getMaplistCross(lat,lon,maplistpattern,georef,scale,True)
```

Also add the new function getMaplistCross() before getForecast()

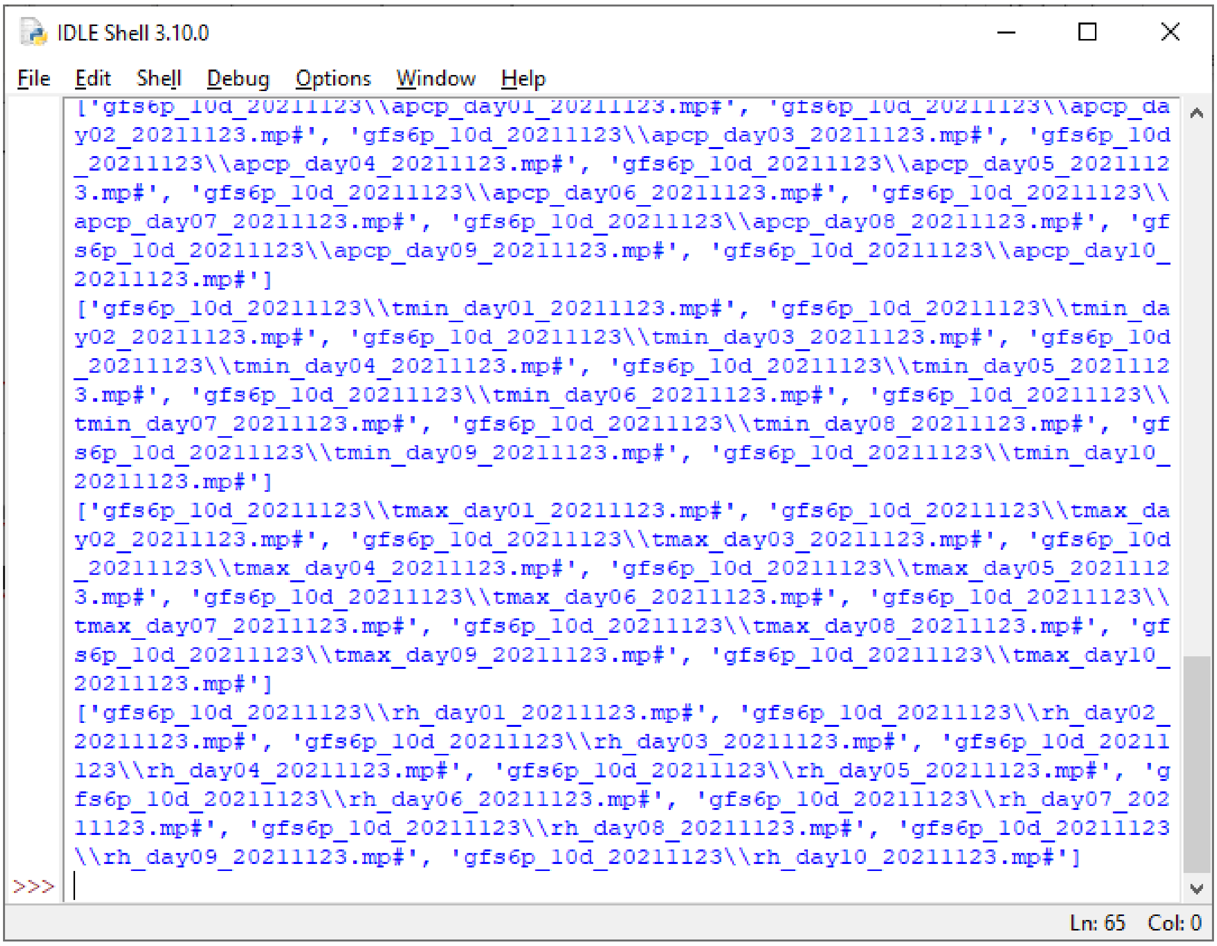
```
def getMaplistCross(lat, lon, maplistpattern, georef, scale, long):

    files = sorted(glob.glob(maplistpattern))

    print(files)
```

Result:



```
def getMaplistCross(lat, lon, maplistpattern, georef, scale, long):
    files = sorted(glob.glob(maplistpattern))
    print(files)
```

Run the script again. For each of the 6 parameters, the filenames of the 10 forecast files are displayed, confirming that the script correctly identifies the source data files.



```
['gfs6p_10d_20211123\\apcp_day01_20211123.mp#', 'gfs6p_10d_20211123\\apcp_da
y02_20211123.mp#', 'gfs6p_10d_20211123\\apcp_day03_20211123.mp#', 'gfs6p_10d
_20211123\\apcp_day04_20211123.mp#', 'gfs6p_10d_20211123\\apcp_day05_2021112
3.mp#', 'gfs6p_10d_20211123\\apcp_day06_20211123.mp#', 'gfs6p_10d_20211123\\
apcp_day07_20211123.mp#', 'gfs6p_10d_20211123\\apcp_day08_20211123.mp#', 'gf
s6p_10d_20211123\\apcp_day09_20211123.mp#', 'gfs6p_10d_20211123\\apcp_day10_
20211123.mp#']
['gfs6p_10d_20211123\\tmin_day01_20211123.mp#', 'gfs6p_10d_20211123\\tmin_da
y02_20211123.mp#', 'gfs6p_10d_20211123\\tmin_day03_20211123.mp#', 'gfs6p_10d
_20211123\\tmin_day04_20211123.mp#', 'gfs6p_10d_20211123\\tmin_day05_2021112
3.mp#', 'gfs6p_10d_20211123\\tmin_day06_20211123.mp#', 'gfs6p_10d_20211123\\
tmin_day07_20211123.mp#', 'gfs6p_10d_20211123\\tmin_day08_20211123.mp#', 'gf
s6p_10d_20211123\\tmin_day09_20211123.mp#', 'gfs6p_10d_20211123\\tmin_day10_
20211123.mp#']
['gfs6p_10d_20211123\\tmax_day01_20211123.mp#', 'gfs6p_10d_20211123\\tmax_da
y02_20211123.mp#', 'gfs6p_10d_20211123\\tmax_day03_20211123.mp#', 'gfs6p_10d
_20211123\\tmax_day04_20211123.mp#', 'gfs6p_10d_20211123\\tmax_day05_2021112
3.mp#', 'gfs6p_10d_20211123\\tmax_day06_20211123.mp#', 'gfs6p_10d_20211123\\
tmax_day07_20211123.mp#', 'gfs6p_10d_20211123\\tmax_day08_20211123.mp#', 'gf
s6p_10d_20211123\\tmax_day09_20211123.mp#', 'gfs6p_10d_20211123\\tmax_day10_
20211123.mp#']
['gfs6p_10d_20211123\\rh_day01_20211123.mp#', 'gfs6p_10d_20211123\\rh_day02_
20211123.mp#', 'gfs6p_10d_20211123\\rh_day03_20211123.mp#', 'gfs6p_10d_20211
123\\rh_day04_20211123.mp#', 'gfs6p_10d_20211123\\rh_day05_20211123.mp#', 'g
fs6p_10d_20211123\\rh_day06_20211123.mp#', 'gfs6p_10d_20211123\\rh_day07_202
11123.mp#', 'gfs6p_10d_20211123\\rh_day08_20211123.mp#', 'gfs6p_10d_20211123
\\rh_day09_20211123.mp#', 'gfs6p_10d_20211123\\rh_day10_20211123.mp#']
>>> 
```

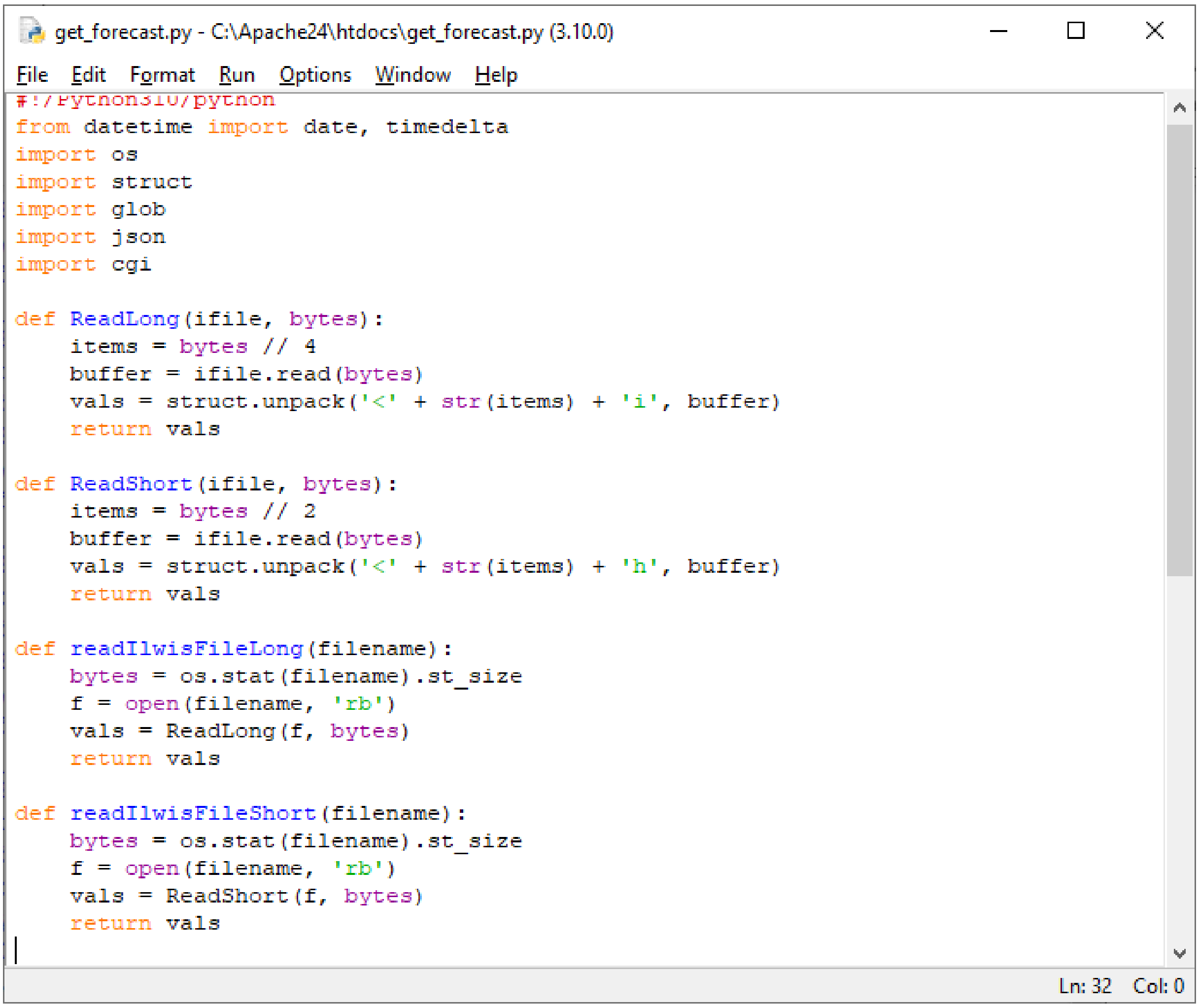
Step 3: read the binary ILWIS files

Add the following code near the top of the file, before getMaplistCross() but after import cgi.

This defines four functions that are able to read the two types of ILWIS binary file (.mp# files) that are available in the GFS zipfile: “short” (2 bytes per pixel) and “long”: 4 bytes per pixel. The entire binary file is read into a python array.

```
def ReadLong(ifile, bytes):
    items = bytes // 4
    buffer = ifile.read(bytes)
    vals = struct.unpack('<' + str(items) + 'i', buffer)
    return vals


def ReadShort(ifile, bytes):
    items = bytes // 2
    buffer = ifile.read(bytes)
    vals = struct.unpack('<' + str(items) + 'h', buffer)
    return vals

def readIlwisFileLong(filename):
    bytes = os.stat(filename).st_size
    f = open(filename, 'rb')
    vals = ReadLong(f, bytes)
    return vals

def readIlwisFileShort(filename):
    bytes = os.stat(filename).st_size
    f = open(filename, 'rb')
    vals = ReadShort(f, bytes)
    return vals
```

Result:

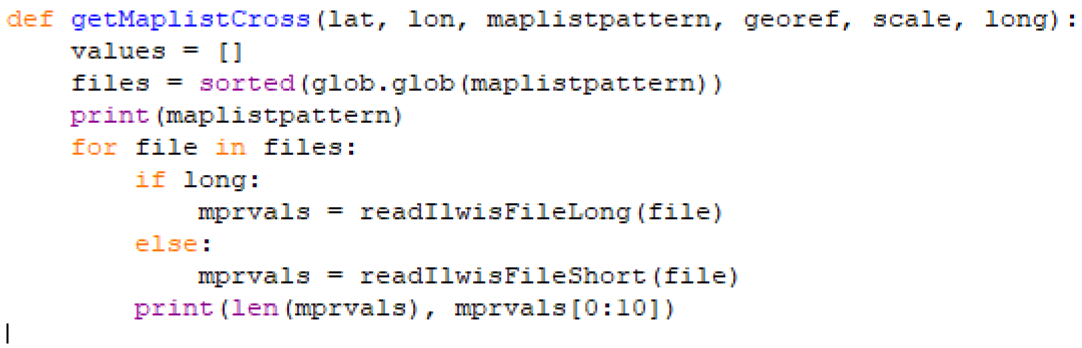

get_forecast.py - C:\Apache24\htdocs\get_forecast.py (3.10.0)
File Edit Format Run Options Window Help
Ln: 32 Col: 0

```
#!/Python310/python
from datetime import date, timedelta
import os
import struct
import glob
import json
import cgi

def ReadLong(ifile, bytes):
    items = bytes // 4
    buffer = ifile.read(bytes)
    vals = struct.unpack('<' + str(items) + 'i', buffer)
    return vals

def ReadShort(ifile, bytes):
    items = bytes // 2
    buffer = ifile.read(bytes)
    vals = struct.unpack('<' + str(items) + 'h', buffer)
    return vals

def readIlwisFileLong(filename):
    bytes = os.stat(filename).st_size
    f = open(filename, 'rb')
    vals = ReadLong(f, bytes)
    return vals

def readIlwisFileShort(filename):
    bytes = os.stat(filename).st_size
    f = open(filename, 'rb')
    vals = ReadShort(f, bytes)
    return vals
```

Replace getMaplistCross() with this version:

```
def getMaplistCross(lat, lon, maplistpattern, georef, scale, long):
    values = []
    files = sorted(glob.glob(maplistpattern))
    print(maplistpattern)
    for file in files:
        if long:
            mprvals = readIlwisFileLong(file)
        else:
            mprvals = readIlwisFileShort(file)
        print(len(mprvals), mprvals[0:10])
```

Result:

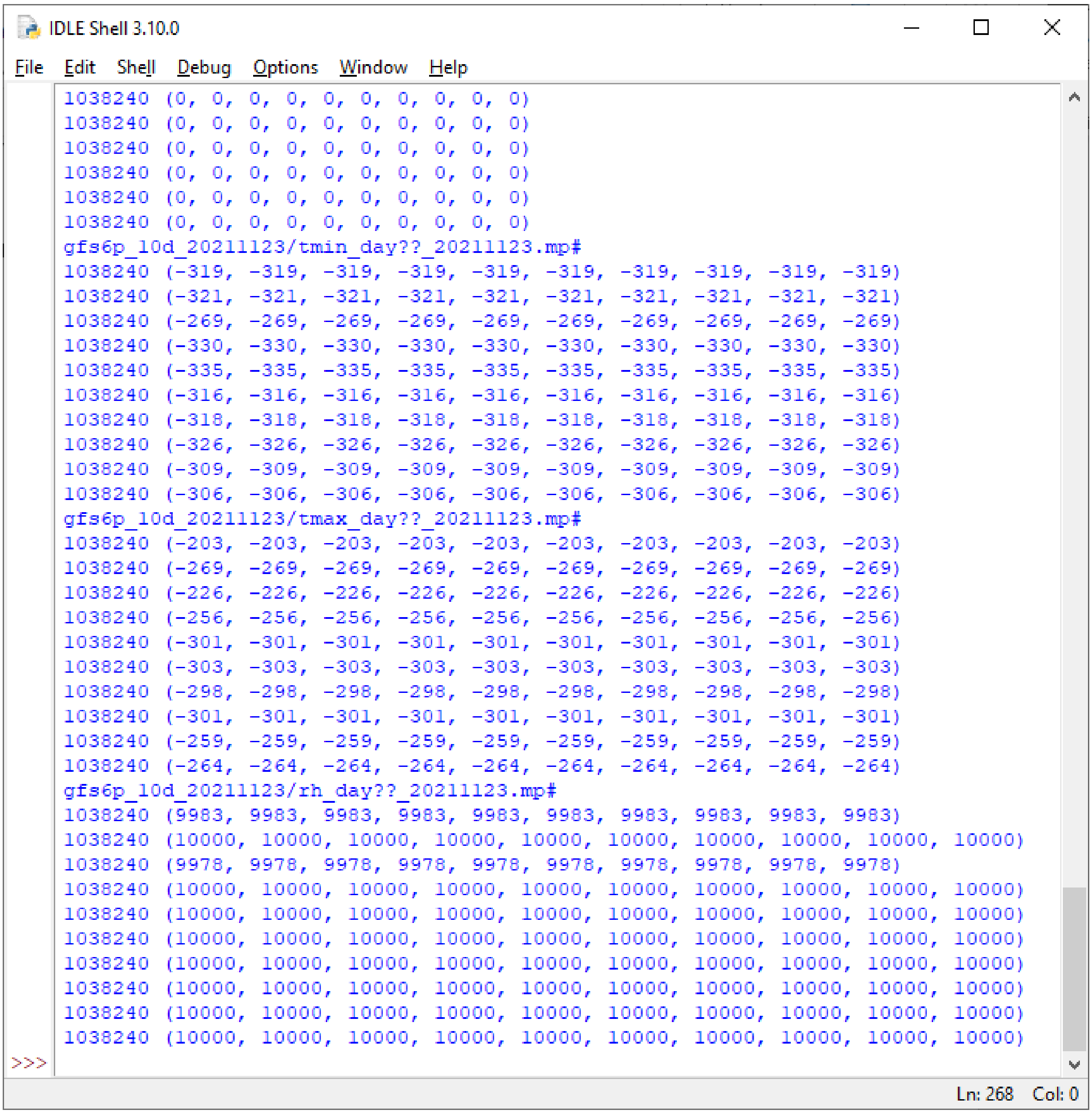


```
def getMaplistCross(lat, lon, maplistpattern, georef, scale, long):
    values = []
    files = sorted(glob.glob(maplistpattern))
    print(maplistpattern)
    for file in files:
        if long:
            mprvals = readIlwisFileLong(file)
        else:
            mprvals = readIlwisFileShort(file)
        print(len(mprvals), mprvals[0:10])
```

|

This now prints the total length of the ILWIS data file that was read, and also the first 10 numbers of every data file.

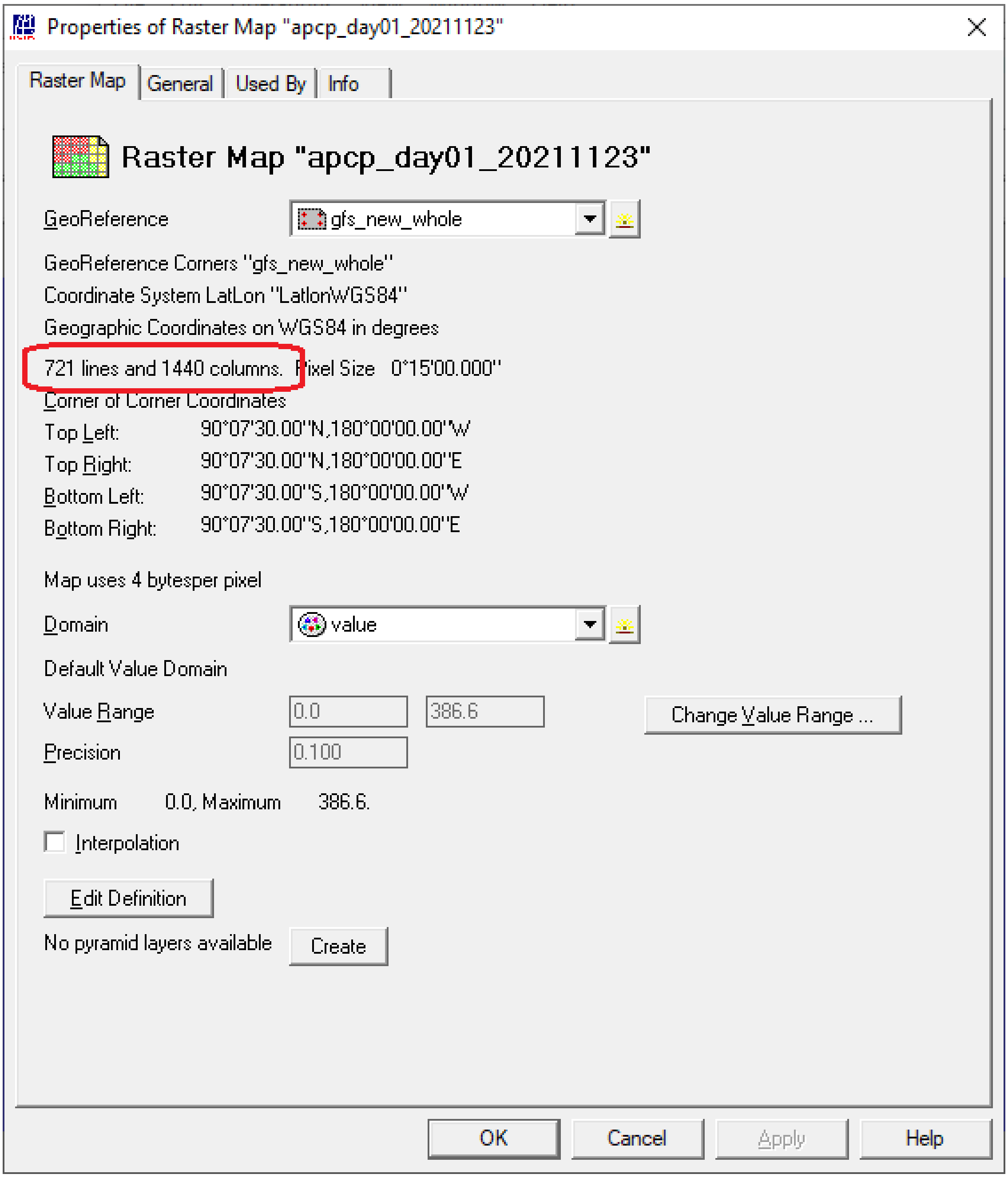
Executing the script will show a result similar to this:



```
1038240 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
1038240 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
1038240 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
1038240 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
1038240 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
1038240 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
gfs6p_10d_20211123/tmin_day??_20211123.mp#
1038240 (-319, -319, -319, -319, -319, -319, -319, -319, -319, -319)
1038240 (-321, -321, -321, -321, -321, -321, -321, -321, -321, -321)
1038240 (-269, -269, -269, -269, -269, -269, -269, -269, -269, -269)
1038240 (-330, -330, -330, -330, -330, -330, -330, -330, -330, -330)
1038240 (-335, -335, -335, -335, -335, -335, -335, -335, -335, -335)
1038240 (-316, -316, -316, -316, -316, -316, -316, -316, -316, -316)
1038240 (-318, -318, -318, -318, -318, -318, -318, -318, -318, -318)
1038240 (-326, -326, -326, -326, -326, -326, -326, -326, -326, -326)
1038240 (-309, -309, -309, -309, -309, -309, -309, -309, -309, -309)
1038240 (-306, -306, -306, -306, -306, -306, -306, -306, -306, -306)
gfs6p_10d_20211123/tmax_day??_20211123.mp#
1038240 (-203, -203, -203, -203, -203, -203, -203, -203, -203, -203)
1038240 (-269, -269, -269, -269, -269, -269, -269, -269, -269, -269)
1038240 (-226, -226, -226, -226, -226, -226, -226, -226, -226, -226)
1038240 (-256, -256, -256, -256, -256, -256, -256, -256, -256, -256)
1038240 (-301, -301, -301, -301, -301, -301, -301, -301, -301, -301)
1038240 (-303, -303, -303, -303, -303, -303, -303, -303, -303, -303)
1038240 (-298, -298, -298, -298, -298, -298, -298, -298, -298, -298)
1038240 (-301, -301, -301, -301, -301, -301, -301, -301, -301, -301)
1038240 (-259, -259, -259, -259, -259, -259, -259, -259, -259, -259)
1038240 (-264, -264, -264, -264, -264, -264, -264, -264, -264, -264)
gfs6p_10d_20211123/rh_day??_20211123.mp#
1038240 (9983, 9983, 9983, 9983, 9983, 9983, 9983, 9983, 9983, 9983)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (9978, 9978, 9978, 9978, 9978, 9978, 9978, 9978, 9978, 9978)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
1038240 (10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)
>>>
```

Step 4: verify the result

Note that the length of all arrays is the same: 1038240 numbers. This corresponds to the dimensions of the raster image: 1440 x 721 pixels; each number is a pixel, stored one-row-at-a-time.



Properties of Raster Map "apcp_day01_20211123"

Raster Map "apcp_day01_20211123"

GeoReference gfs_new_whole

GeoReference Corners "gfs_new_whole"

Coordinate System LatLon "LatlonWGS84"

Geographic Coordinates on WGS84 in degrees

721 lines and 1440 columns. Pixel Size 0°15'00.000"

Corner of Corner Coordinates

Top Left:	90°07'30.00"N,180°00'00.00"W
Top Right:	90°07'30.00"N,180°00'00.00"E
Bottom Left:	90°07'30.00"S,180°00'00.00"W
Bottom Right:	90°07'30.00"S,180°00'00.00"E

Map uses 4 bytesper pixel

Domain value

Default Value Domain

Value Range 0.0 386.6

Precision 0.100

Minimum 0.0, Maximum 386.6.

□ Interpolation

No pyramid layers available

Step 5: read all timeseries values at location lat=0, lon=0

Add the following code above getMaplistCross().

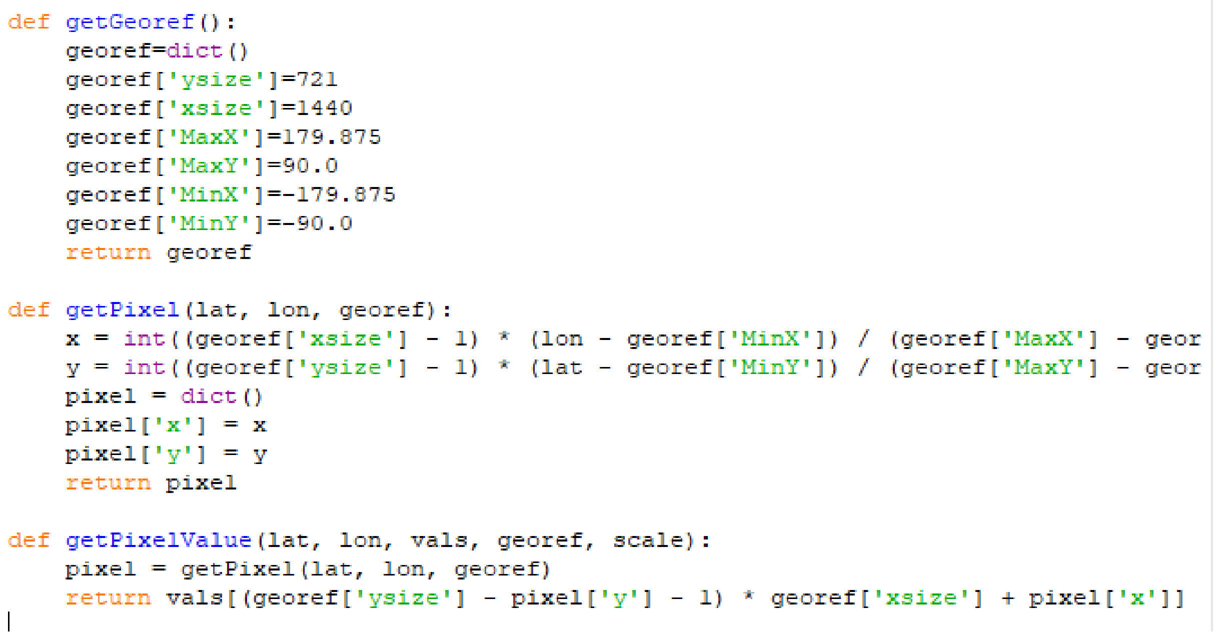
The first function getGeoref() is the actual georeference of the images, without reading any of the metadata files. We know the numbers, so we use them. This is bad practice (officially you would have to read these numbers from the metadata files), but it works for now.

The second function getPixel() computes the actual pixel position (x, y location in a 2D array) given a latitude and longitude.

The third function getPixelValue() computes the pixel position in the 1D array that we have available, and returns the value at that position in the array. The value is scaled before being returned, in order to get the value into the actual unit of the parameter (mm/day, degrees Celsius or percentage). Officially the scaling number should have been read from the metadata files, but is hardcoded here for ease (the scale numbers are in function getForecast()).

```
def getGeoref():
    georef=dict()
    georef['ysize']=721
    georef['xsize']=1440
    georef['MaxX']=179.875
    georef['MaxY']=90.0
    georef['MinX']=-179.875
    georef['MinY']=-90.0
    return georef


def getPixel(lat, lon, georef):
    x = int((georef['xsize'] - 1) * (lon - georef['MinX']) / 
(georef['MaxX'] - georef['MinX']))
    y = int((georef['ysize'] - 1) * (lat - georef['MinY']) / 
(georef['MaxY'] - georef['MinY']))
    pixel = dict()
    pixel['x'] = x
    pixel['y'] = y
    return pixel


def getPixelValue(lat, lon, vals, georef, scale):
    pixel = getPixel(lat, lon, georef)
    return vals[(georef['ysize'] - pixel['y'] - 1) * georef['xsize'] 
+ pixel['x']] / scale
```

The result looks like this:

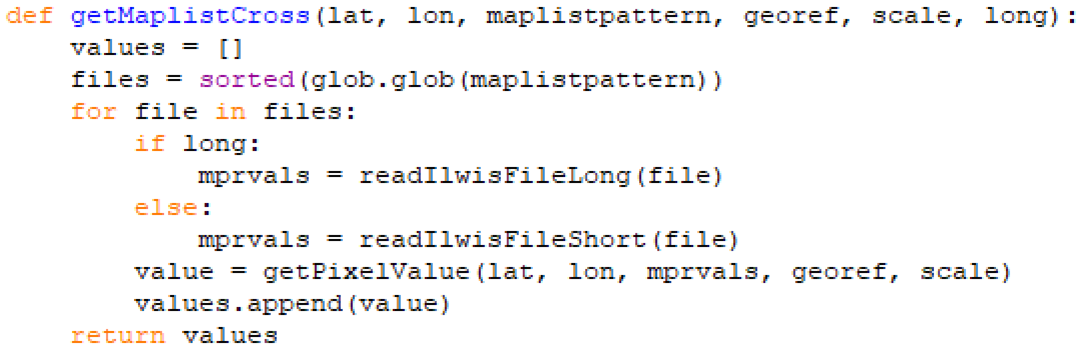


```
def getGeoref():
    georef=dict()
    georef['ysize']=721
    georef['xsize']=1440
    georef['MaxX']=179.875
    georef['MaxY']=90.0
    georef['MinX']=-179.875
    georef['MinY']=-90.0
    return georef

def getPixel(lat, lon, georef):
    x = int((georef['xsize'] - 1) * (lon - georef['MinX']) / (georef['MaxX'] - geor
    y = int((georef['ysize'] - 1) * (lat - georef['MinY']) / (georef['MaxY'] - geor
    pixel = dict()
    pixel['x'] = x
    pixel['y'] = y
    return pixel

def getPixelValue(lat, lon, vals, georef, scale):
    pixel = getPixel(lat, lon, georef)
    return vals[(georef['ysize'] - pixel['y'] - 1) * georef['xsize'] + pixel['x']]
```

Replace once again getMaplistCross() with its final version:

```
def getMaplistCross(lat, lon, maplistpattern, georef, scale, long):

    values = []

    files = sorted(glob.glob(maplistpattern))

    for file in files:

        if long:

            mprvals = readIlwisFileLong(file)

        else:

            mprvals = readIlwisFileShort(file)

        value = getPixelValue(lat, lon, mprvals, georef, scale)

        values.append(value)

    return values
```

Result:



```
def getMaplistCross(lat, lon, maplistpattern, georef, scale, long):
    values = []
    files = sorted(glob.glob(maplistpattern))
    for file in files:
        if long:
            mprvals = readIlwisFileLong(file)
        else:
            mprvals = readIlwisFileShort(file)
        value = getPixelValue(lat, lon, mprvals, georef, scale)
        values.append(value)
    return values
```

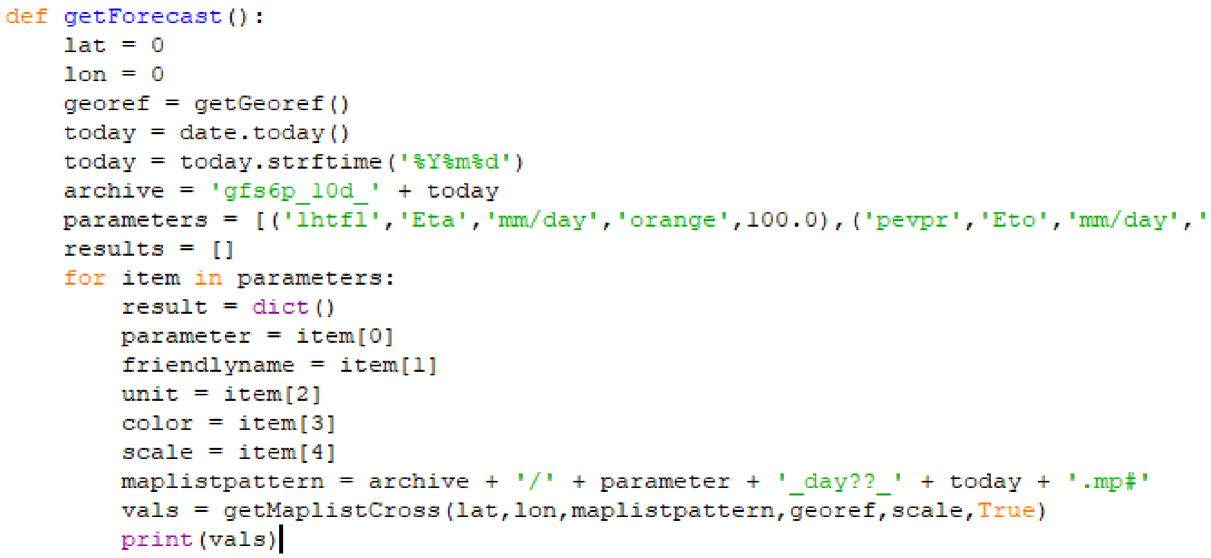
Change function getForecast():

Replace the line georef = None by the call to the newly created georeferenced function:

```
georef = getGeoref()
```

Also add the statement `print(vals)` at the end of getForecast()

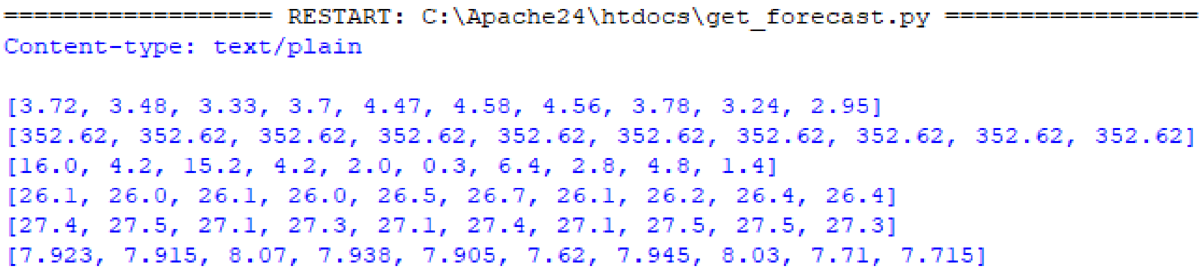
Result:



```
def getForecast():
    lat = 0
    lon = 0
    georef = getGeoref()
    today = date.today()
    today = today.strftime('%Y%m%d')
    archive = 'gfs6p_10d_' + today
    parameters = [('lhtfl','Eta','mm/day','orange',100.0),('pevpr','Eto','mm/day','
    results = []
    for item in parameters:
        result = dict()
        parameter = item[0]
        friendlyname = item[1]
        unit = item[2]
        color = item[3]
        scale = item[4]
        maplistpattern = archive + '/' + parameter + '_day??_' + today + '.mp#'
        vals = getMaplistCross(lat,lon,maplistpattern,georef,scale,True)
        print(vals)
```

Run the script. The result will be 6 arrays with 10 numbers each.

Note that this is at hardcoded location lat = 0 and lon = 0 (Atlantic ocean).



```
================== RESTART: C:\Apache24\htdocs\get_forecast.py =================
Content-type: text/plain

[3.72, 3.48, 3.33, 3.7, 4.47, 4.58, 4.56, 3.78, 3.24, 2.95]
[352.62, 352.62, 352.62, 352.62, 352.62, 352.62, 352.62, 352.62, 352.62, 352.62]
[16.0, 4.2, 15.2, 4.2, 2.0, 0.3, 6.4, 2.8, 4.8, 1.4]
[26.1, 26.0, 26.1, 26.0, 26.5, 26.7, 26.1, 26.2, 26.4, 26.4]
[27.4, 27.5, 27.1, 27.3, 27.1, 27.4, 27.1, 27.5, 27.5, 27.3]
[7.923, 7.915, 8.07, 7.938, 7.905, 7.62, 7.945, 8.03, 7.71, 7.715]
```

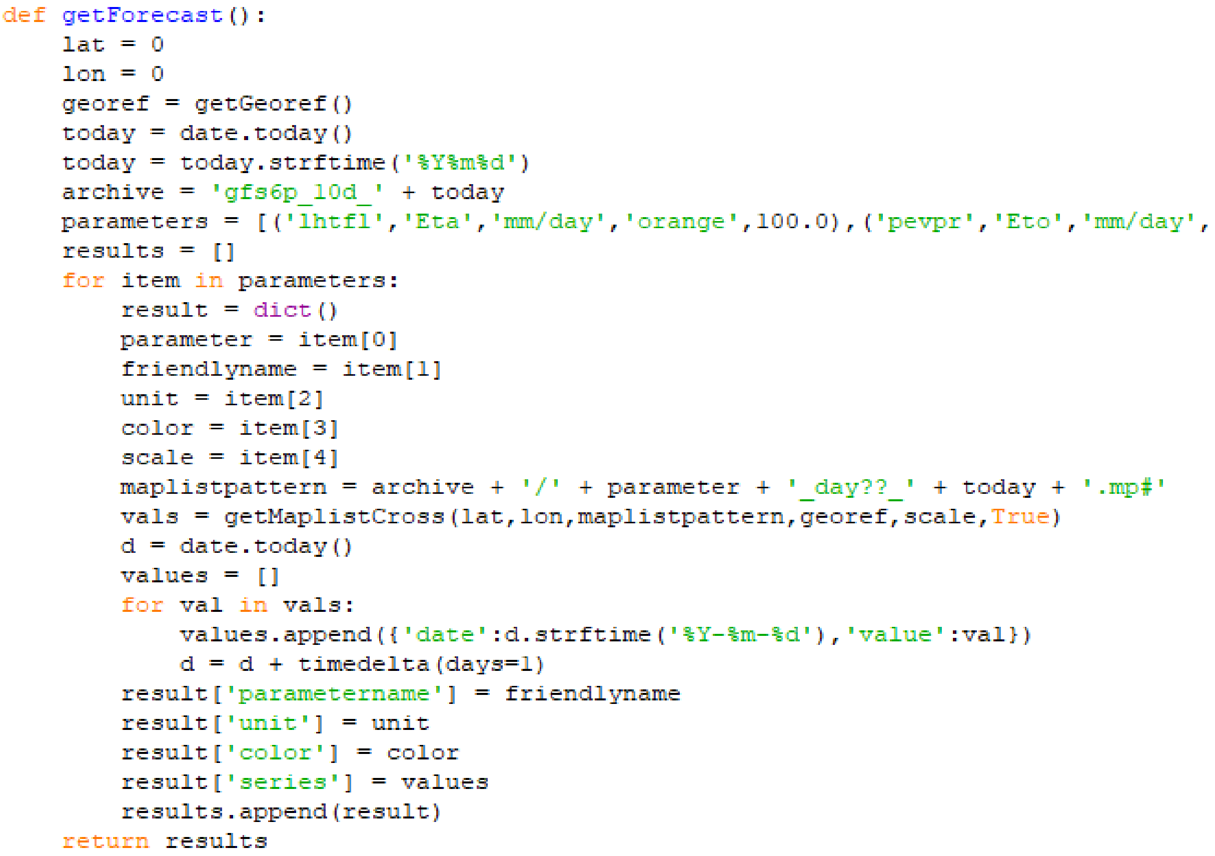
Step 6: add more metadata to the output

Finally, complete the implementation of getForecast(), by producing a python dictionary, containing for each of the parameters the name, the unit, the color (for repeatedly giving the same color when plotting the parameter's graph), and the numbers as date+value pairs.

To do this, delete the print(vals) line, and append the following code at the end of getForecast():

```
        d = date.today()

        values = []

        for val in vals:

            values.append({'date':d.strftime('%Y-%m-%d'),'value':val})

            d = d + timedelta(days=1)
        result['parametername'] = friendlyname
        result['unit'] = unit
        result['color'] = color
        result['series'] = values
        results.append(result)
    return results
```

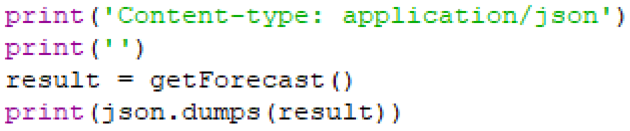
Result:



```
def getForecast():
    lat = 0
    lon = 0
    georef = getGeoref()
    today = date.today()
    today = today.strftime('%Y%m%d')
    archive = 'gfs6p_10d_' + today
    parameters = [('lhtfl','Eta','mm/day','orange',100.0),('pevpr','Eto','mm/day',
    results = []
    for item in parameters:
        result = dict()
        parameter = item[0]
        friendlyname = item[1]
        unit = item[2]
        color = item[3]
        scale = item[4]
        maplistpattern = archive + '/' + parameter + '_day??_' + today + '.mp#'
        vals = getMaplistCross(lat,lon,maplistpattern,georef,scale,True)
        d = date.today()
        values = []
        for val in vals:
            values.append({'date':d.strftime('%Y-%m-%d'),'value':val})
            d = d + timedelta(days=1)
        result['parametername'] = friendlyname
        result['unit'] = unit
        result['color'] = color
        result['series'] = values
        results.append(result)
    return results
```

Replace the main program with the following:

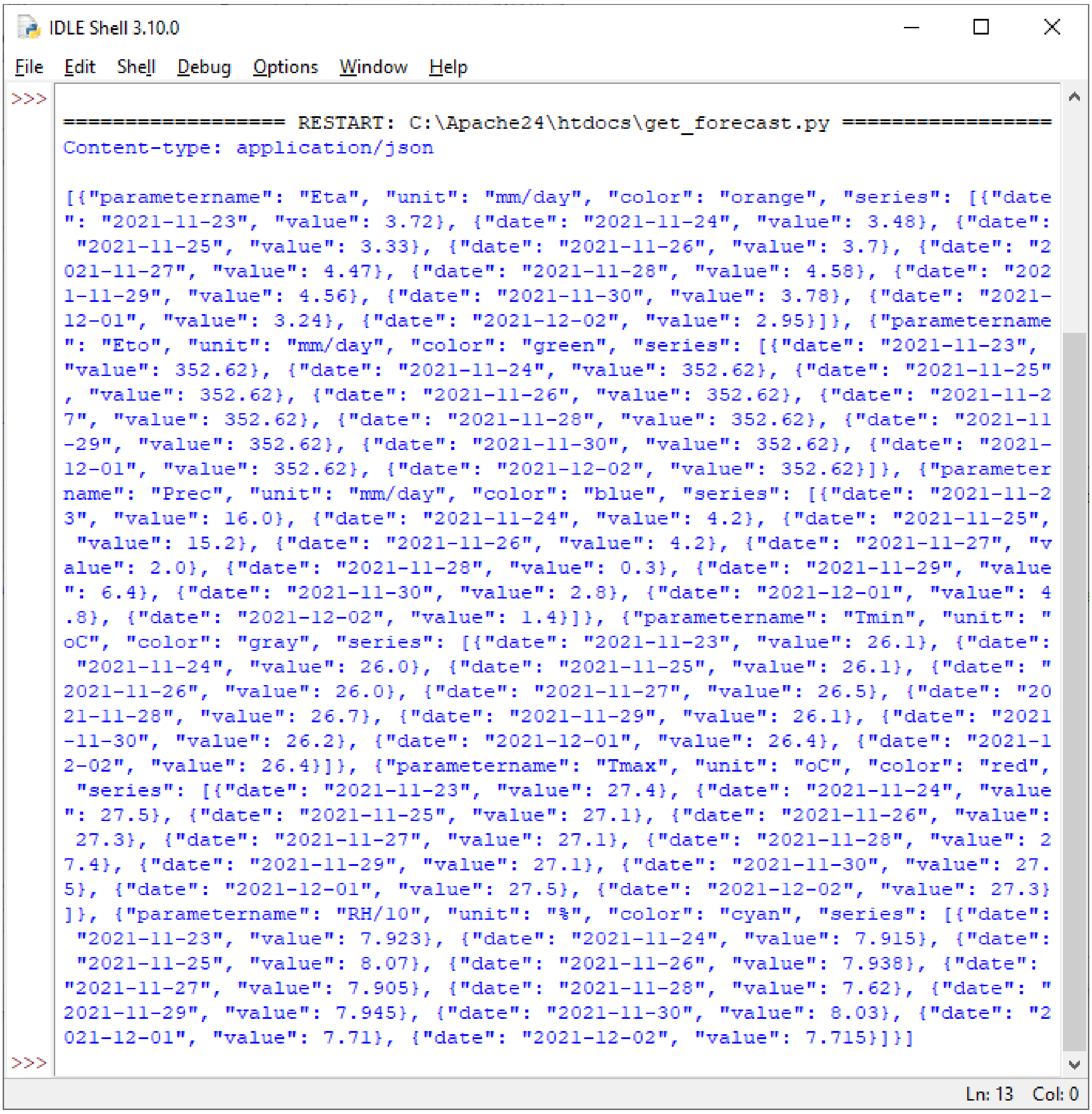
```
print('Content-type: application/json')
print('')
result = getForecast()
print(json.dumps(result))
```



```
print('Content-type: application/json')
print('')
result = getForecast()
print(json.dumps(result))
```

Note that the json.dumps() is to properly format the Python dictionary to the JSON format. Also the Content-type: application/json is to let the client application “know” what the result is.

Run the program.



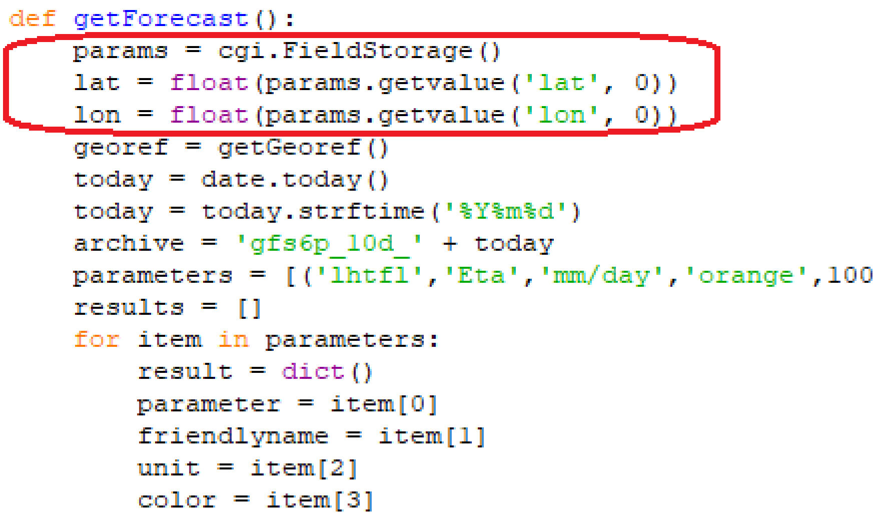
```
>>>
================== RESTART: C:\Apache24\htdocs\get_forecast.py =================
Content-type: application/json

[{"parametername": "Eta", "unit": "mm/day", "color": "orange", "series": [{"date
": "2021-11-23", "value": 3.72}, {"date": "2021-11-24", "value": 3.48}, {"date":
 "2021-11-25", "value": 3.33}, {"date": "2021-11-26", "value": 3.7}, {"date": "2
021-11-27", "value": 4.47}, {"date": "2021-11-28", "value": 4.58}, {"date": "202
1-11-29", "value": 4.56}, {"date": "2021-11-30", "value": 3.78}, {"date": "2021-
12-01", "value": 3.24}, {"date": "2021-12-02", "value": 2.95}]}, {"parametername
": "Eto", "unit": "mm/day", "color": "green", "series": [{"date": "2021-11-23",
"value": 352.62}, {"date": "2021-11-24", "value": 352.62}, {"date": "2021-11-25"
, "value": 352.62}, {"date": "2021-11-26", "value": 352.62}, {"date": "2021-11-2
7", "value": 352.62}, {"date": "2021-11-28", "value": 352.62}, {"date": "2021-11
-29", "value": 352.62}, {"date": "2021-11-30", "value": 352.62}, {"date": "2021-
12-01", "value": 352.62}, {"date": "2021-12-02", "value": 352.62}]}, {"parameter
name": "Prec", "unit": "mm/day", "color": "blue", "series": [{"date": "2021-11-2
3", "value": 16.0}, {"date": "2021-11-24", "value": 4.2}, {"date": "2021-11-25",
 "value": 15.2}, {"date": "2021-11-26", "value": 4.2}, {"date": "2021-11-27", "v
alue": 2.0}, {"date": "2021-11-28", "value": 0.3}, {"date": "2021-11-29", "value
": 6.4}, {"date": "2021-11-30", "value": 2.8}, {"date": "2021-12-01", "value": 4
.8}, {"date": "2021-12-02", "value": 1.4}]}, {"parametername": "Tmin", "unit": "
oC", "color": "gray", "series": [{"date": "2021-11-23", "value": 26.1}, {"date":
 "2021-11-24", "value": 26.0}, {"date": "2021-11-25", "value": 26.1}, {"date": "
2021-11-26", "value": 26.0}, {"date": "2021-11-27", "value": 26.5}, {"date": "20
21-11-28", "value": 26.7}, {"date": "2021-11-29", "value": 26.1}, {"date": "2021
-11-30", "value": 26.2}, {"date": "2021-12-01", "value": 26.4}, {"date": "2021-1
2-02", "value": 26.4}]}, {"parametername": "Tmax", "unit": "oC", "color": "red",
 "series": [{"date": "2021-11-23", "value": 27.4}, {"date": "2021-11-24", "value
": 27.5}, {"date": "2021-11-25", "value": 27.1}, {"date": "2021-11-26", "value":
 27.3}, {"date": "2021-11-27", "value": 27.1}, {"date": "2021-11-28", "value": 2
7.4}, {"date": "2021-11-29", "value": 27.1}, {"date": "2021-11-30", "value": 27.
5}, {"date": "2021-12-01", "value": 27.5}, {"date": "2021-12-02", "value": 27.3}
]}, {"parametername": "RH/10", "unit": "%", "color": "cyan", "series": [{"date":
 "2021-11-23", "value": 7.923}, {"date": "2021-11-24", "value": 7.915}, {"date":
 "2021-11-25", "value": 8.07}, {"date": "2021-11-26", "value": 7.938}, {"date":
"2021-11-27", "value": 7.905}, {"date": "2021-11-28", "value": 7.62}, {"date": "
2021-11-29", "value": 7.945}, {"date": "2021-11-30", "value": 8.03}, {"date": "2
021-12-01", "value": 7.71}, {"date": "2021-12-02", "value": 7.715}]}]
>>>
```

Now the result is the required JSON output.

Step 7: make the script get the lat/lon from the parameters of the webserver

In order to let the program capture the latitude and longitude from the parameters of the URL, add the following 3 lines to the beginning of the getForecast() function (replace the lines with lat = 0 and lon = 0):

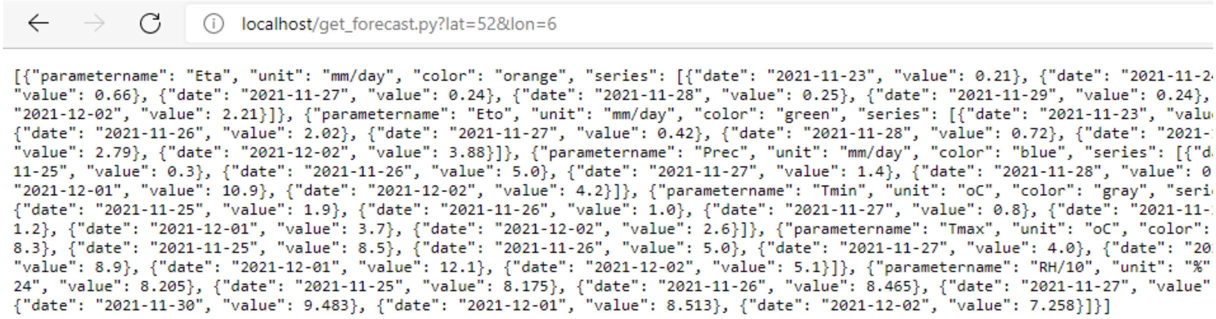
```
params = cgi.FieldStorage()

lat = float(params.getvalue('lat', 0))

lon = float(params.getvalue('lon', 0))
```



```
def getForecast():
    params = cgi.FieldStorage()
    lat = float(params.getvalue('lat', 0))
    lon = float(params.getvalue('lon', 0))
    georef = getGeoref()
    today = date.today()
    today = today.strftime('%Y%m%d')
    archive = 'gfs6p_10d_' + today
    parameters = [('lhtfl','Eta','mm/day','orange',100
    results = []
    for item in parameters:
        result = dict()
        parameter = item[0]
        friendlyname = item[1]
        unit = item[2]
        color = item[3]
```

The program can now be run from the browser, and is perfectly suited to serve as the Web API function that returns the GFS6p forecast values given a latitude and longitude:

localhost/get_forecast.py?lat=52&lon=6



localhost/get_forecast.py?lat=52&lon=6

```
[{"parametername": "Eta", "unit": "mm/day", "color": "orange", "series": [{"date": "2021-11-23", "value": 0.21}, {"date": "2021-11-2
"value": 0.66}, {"date": "2021-11-27", "value": 0.24}, {"date": "2021-11-28", "value": 0.25}, {"date": "2021-11-29", "value": 0.24},
"2021-12-02", "value": 2.21}]}, {"parametername": "Eto", "unit": "mm/day", "color": "green", "series": [{"date": "2021-11-23", "valu
{"date": "2021-11-26", "value": 2.02}, {"date": "2021-11-27", "value": 0.42}, {"date": "2021-11-28", "value": 0.72}, {"date": "2021-
"value": 2.79}, {"date": "2021-12-02", "value": 3.88}]}, {"parametername": "Prec", "unit": "mm/day", "color": "blue", "series": [{"d
11-25", "value": 0.3}, {"date": "2021-11-26", "value": 5.0}, {"date": "2021-11-27", "value": 1.4}, {"date": "2021-11-28", "value": 0
"2021-12-01", "value": 10.9}, {"date": "2021-12-02", "value": 4.2}]}, {"parametername": "Tmin", "unit": "oC", "color": "gray", "seri
{"date": "2021-11-25", "value": 1.9}, {"date": "2021-11-26", "value": 1.0}, {"date": "2021-11-27", "value": 0.8}, {"date": "2021-11-
1.2}, {"date": "2021-12-01", "value": 3.7}, {"date": "2021-12-02", "value": 2.6}]}, {"parametername": "Tmax", "unit": "oC", "color":
8.3}, {"date": "2021-11-25", "value": 8.5}, {"date": "2021-11-26", "value": 5.0}, {"date": "2021-11-27", "value": 4.0}, {"date": "20
"value": 8.9}, {"date": "2021-12-01", "value": 12.1}, {"date": "2021-12-02", "value": 5.1}]}, {"parametername": "RH/10", "unit": "%"
24", "value": 8.205}, {"date": "2021-11-25", "value": 8.175}, {"date": "2021-11-26", "value": 8.465}, {"date": "2021-11-27", "value"
{"date": "2021-11-30", "value": 9.483}, {"date": "2021-12-01", "value": 8.513}, {"date": "2021-12-02", "value": 7.258}]}]
```

Official Web API for GFS6p at the ITC

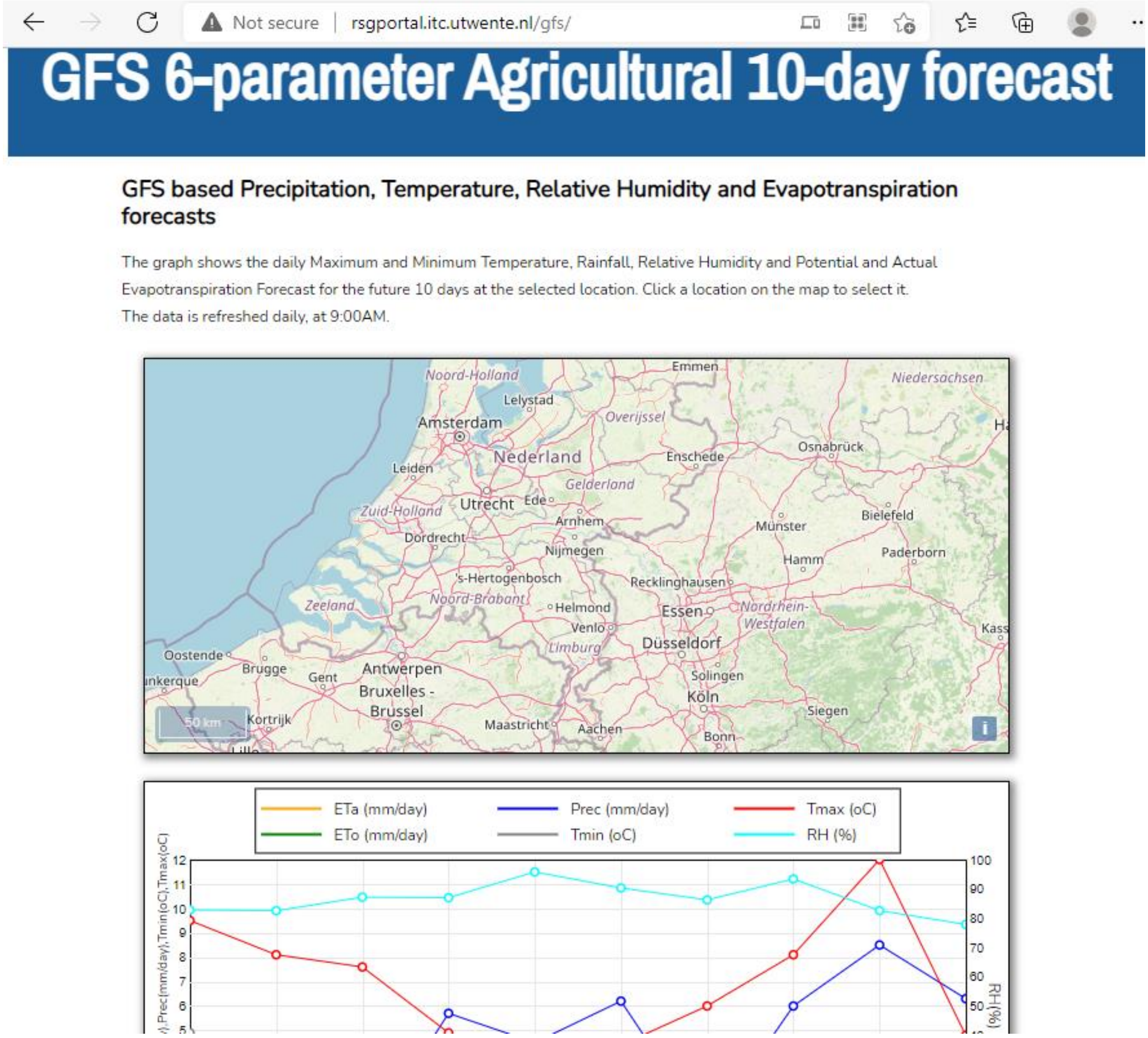
The official Web API that we have installed at the ITC for GFS6p is hosted at rsgportal.itc.utwente.nl/gfs:

rsgportal.itc.utwente.nl/gfs/get_forecast_ext.py?lat=52&lon=6

Note that due to the nature of dictionaries, the order of the name/value pairs returned is random, so if you open the same URL multiple times, the result is not identical (the data is the same, just in a different order).

Note also that your self-created script takes about 1 second to compute and return the results, while the official script has been optimized to be faster (about 0.1 second). This additional speed-optimization effort will be appreciated later on when creating the client software that consumes this Web API service.

A web-client is available here: http://rsgportal.itc.utwente.nl/gfs/



GFS 6-parameter Agricultural 10-day forecast

GFS based Precipitation, Temperature, Relative Humidity and Evapotranspiration forecasts

The graph shows the daily Maximum and Minimum Temperature, Rainfall, Relative Humidity and Potential and Actual Evapotranspiration Forecast for the future 10 days at the selected location. Click a location on the map to select it.
The data is refreshed daily, at 9:00AM.

Questions

What can we do to further speed-optimize the code that we created in get_forecast.py?

What other improvements can you think of? What e.g. happens after midnight? Or what happens when the scheduled task to fetch the zipfile runs before the zipfile was actually in-place?