How to Python Web Scrape the Nasdaq Stock Ex-Dividend Calendar

Web scrape a list of stocks with upcoming ex-dividend dates from the Nasdaq API using the Python pandas, requests, datetime, and calendar modules.

Debra Ray Follow
Jan 19 · 9 min read ★

Photo by Estée Janssens on Unsplash

Install & import required Python packages

You will need the pandas, requests, datetime, and calendar modules to web scrape a list of stocks with upcoming ex-dividend dates from the Nasdaq API using the Python coding language.

```
import pandas, requests, datetime, calendar
```

- **Pandas:** A data analysis library used to create and manipulate data within structures called dataframes. Learn more about the pandas module.
- **Requests:** An HTTP library for Python with a built-in JSON decoder. Learn more about the requests module.
- **Datetime:** Functions to create and format date and time objects in Python. Learn more about the datetime module.
- **Calendar:** Contains a method to retrieve the number of days in a month given a year. Learn more about the calendar module.

Object-oriented programming: Create a calendar class

Object-oriented programming is a great way to organize Python code. There are four primary principals of object-oriented programming: **encapsulation**, **abstraction**, **inheritance**, and **polymorphism.**

- **Encapsulation:** Internal functions called methods built into the class can only be used within that class.
- **Abstraction:** Templates act as blueprints for classes and methods in programs to reduce the time it takes to name, learn, and write code.
- **Inheritance:** Classes can inherit attributes from each other to share common behaviours while maintaining unique features.
- **Polymorphism:** Override inherited parent class methods to give specific behaviours to the child class.

In our example, we want to create an object containing a calendar. By creating a class, we can assign functions called methods to the object that cannot be used outside of the class.

We can also define attributes with variable values. In other words, we can create multiple calendar objects for every month of the year and assign different values to the object's attributes.

Instance attributes versus class attributes

The __init__ function is a constructor function that contains **instance attributes**. You can also define **class attributes** outside of the constructor function. Class attributes are shared among every instance of an object class whereas instance attributes can vary between objects.

```
class dividend_calendar:
     #class attributes
     calendars = []
     url = 'https://api.nasdaq.com/api/calendar/dividends'
     hdrs =  {'Accept': 'application/json, text/plain, */*',
                 'DNT': "1",
                 'Origin': 'https://www.nasdaq.com/',
                 'Sec-Fetch-Mode': 'cors',
                 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0)'}

     def __init__(self, year, month):
          '''
        Parameters
        ----------
        year : year int
        month : month int

        Returns
        -------
        Sets instance attributes for year and month of object.

          '''
          #instance attributes
          self.year = int(year)
          self.month = int(month)
```

For every instance of a class that we make within a kernel, the objects will all possess the same values for any class attributes. For example, we have assigned three class attributes: calendars, url, and hdrs.

The dividend class also possesses two instance attributes: year and month. One assigns values to these attributes at the object level when initiating the class. Thus, each instance of a class can have different values for these attributes.

Web scraping JSON files with Python

The **requests module** has a built-in method to decode JSON (JavaScript Object Notation) dictionaries from a URL. Scraping information from JSON objects can be faster and more convenient than parsing HTML data using XPaths.

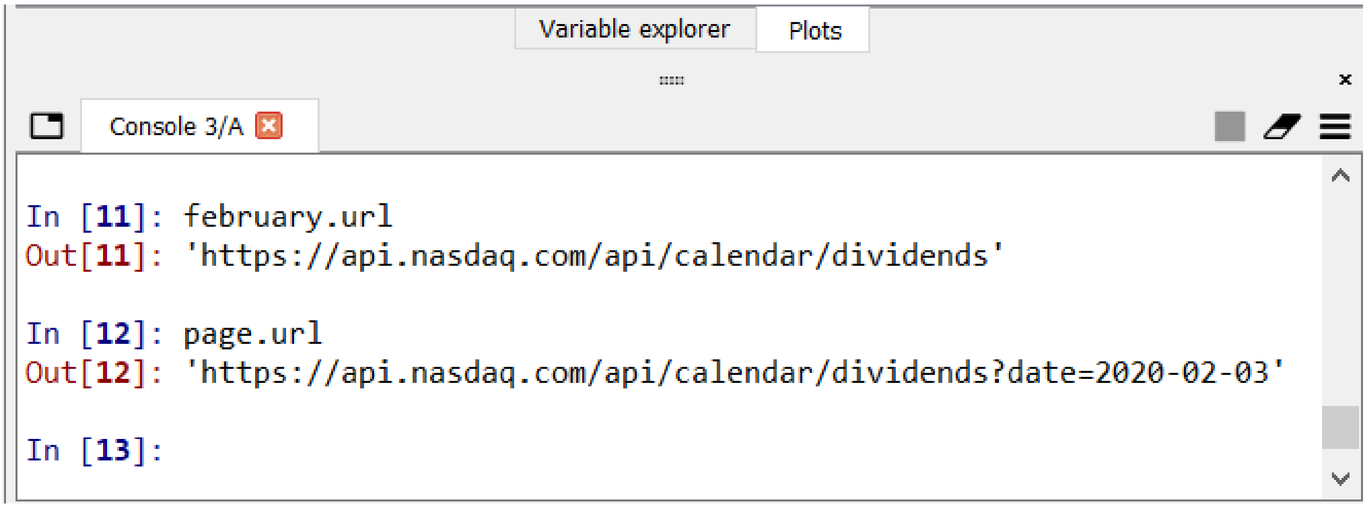
```
def scraper(self, date_str):
         '''
          Scrapes JSON object from page using requests module

           Parameters
           - - - - -
           url : URL string
           hdrs : Header information
           date_str: string in yyyy-mm-dd format

           Returns
           - - - -
           dictionary : Returns a JSON dictionary at a given URL.

         '''
         params = {'date': date_str}
         page=requests.get(self.url,headers=self.hdrs,params=params)
         dictionary = page.json()
         return dictionary
```

To pull the JSON dictionary using the requests module, we must first send a GET request to the server with three arguments: URL, headers, and params. The GET request should then yield a requests object containing a JSON which you can easily decode by calling the `json` method on the returned request.

```
page=requests.get(self.url,headers=self.hdrs,params=params)
dictionary = page.json()
```

The **params argument** should contain API key-value pairs or queries that we want to pass into the URL. For the Nasdaq API, to retrieve the ex-dividend calendar for a particular day, we must pass in a string value for the date key in the format YYYY-MM-DD: `params = {'date': date_str}.`

The URL is defined above the __init__ function as a class attribute with a static string value: `url = 'https://api.nasdaq.com/api/calendar/dividends'` . The `requests.get` method will use the dictionary `params = {'date': date_str}` passed into the params argument to complete the URL by appending a question mark followed by the key, `date=`, and then the value of the `date_str` .



```
In [11]: february.url
Out[11]: 'https://api.nasdaq.com/api/calendar/dividends'

In [12]: page.url
Out[12]: 'https://api.nasdaq.com/api/calendar/dividends?date=2020-02-03'

In [13]:
```

Params argument in the `requests.get` method appends the key-value pair to the URL.

Create & format the date string using the datetime module

The **datetime module** makes it easy to create datetime objects which you can format into strings using the `strftime` method:

```
def date_str(self, day):
        date_obj = datetime.date(self.year, self.month, day)
        date_str = date_obj.strftime(format='%Y-%m-%d')
        return date_str
```

The format code `'%Y-%m-%d'` will format the date into a string containing a four-character year, a two-character month, and a two-character day separated by dashes: YYYY-MM-DD.

Convert JSON dictionary into pandas dataframe

The last method takes the JSON dictionary and converts it to a pandas dataframe. It also appends the pandas dataframe to the calendars list that we created as a class attribute.

```
def dict_to_df(self, dicti):
         '''
         Converts the JSON dictionary into a pandas dataframe
         Appends the dataframe to calendars class attribute

         Parameters
         ----------
         dicti : Output from the scraper method as input.

         Returns
         -------
         calendar : Dataframe of stocks with that exdividend date

         Will append a dataframe to the calendars list (class
         attribute). Otherwise, it will return an empty dataframe.
         '''

         rows = dicti.get('data').get('calendar').get('rows')
         calendar = pandas.DataFrame(rows)
         self.calendars.append(calendar) #append df to calendars
         return calendar
```

Every time we run this method, it will save the dataframe to the **calendars attribute.** We will be able to retrieve it from any instance of the `dividend_calendar` class since it is a class attribute versus an instance attribute.

Moreover, programmers can avoid exceptions interrupting their programs by using the built-in `get` method on the dictionary object: `rows = dicti.get('data').get('calendar').get('rows')`. If the returned JSON object does not possess a data, calendar, or row index, the program will return an empty dataframe.

On the other hand, calling indices instead such as `rows = dicti['data']['calendar']['rows']` is more likely to result in a program interruption. If the JSON indices are an exception from the standard, the code will cease to function and return an error message. We want to avoid this by using the `get` method.

Combine functions into one method

Combining the various functions we created above into one method will make the program more user-friendly.

```
def calendar(self, day):
     '''
     Combines the date_str, scraper, and dict_to_df methods

     Parameters
     ----------
     day : day of the month as string or number.

     Returns
     -------
     dictionary : Returns a JSON dictionary with keys
     dictionary.keys() => data, message, status

     Next Levels:
     dictionary['data'].keys() => calendar, timeframe
     dictionary['data']['calendar'].keys() => headers, rows
     dictionary['data']['calendar']['headers'] => column names
     dictionary['data']['calendar']['rows'] => dictionary list

     '''
     day = int(day)
     date_str = self.date_str(day)
     dictionary = self.scraper(date_str)
     self.dict_to_df(dictionary)
     return dictionary
```

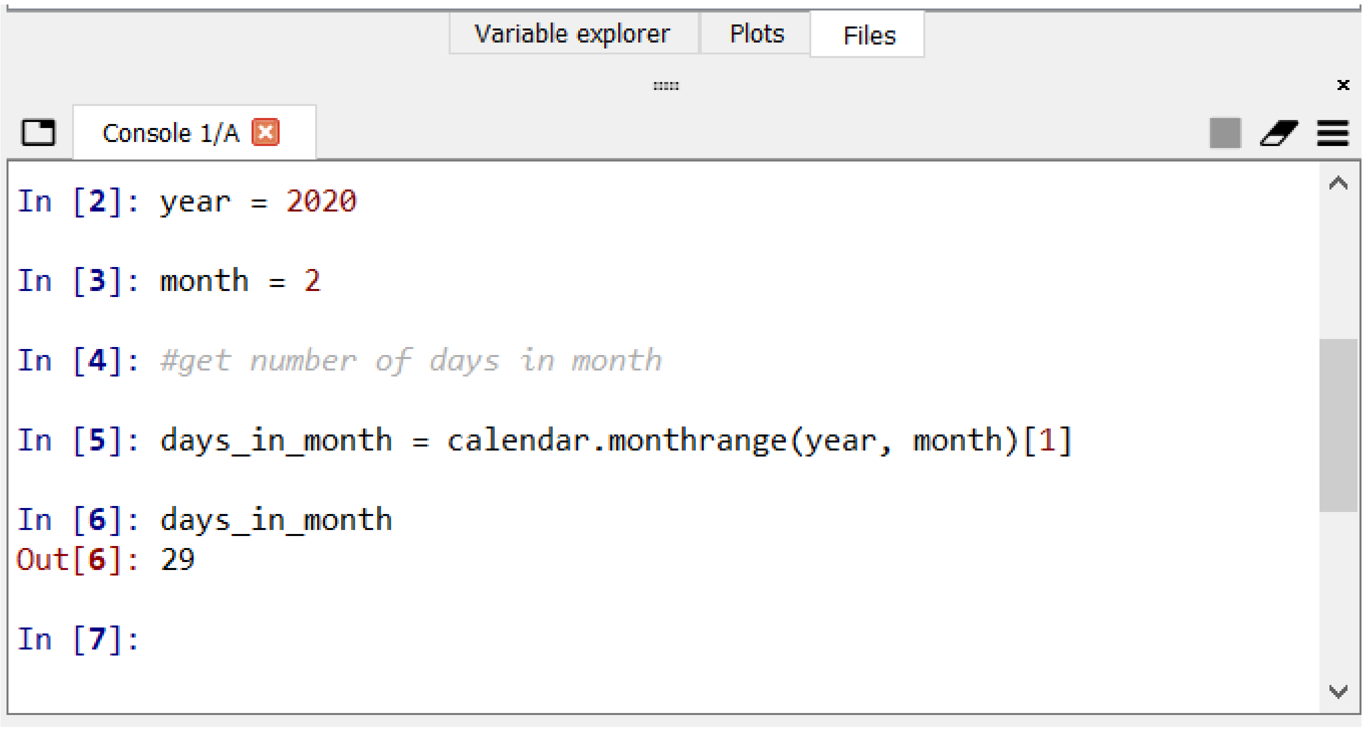
How to use the Python program to scrape the Nasdaq API

Step 1: *Get the number of days in a month.* At the beginning of the program, we imported the **calendar module** which is a collection of Python functions that allow us to easily retrieve calendar information.

The `monthrange` method of the calendar module to quickly lookup how many days there are within one month. This method will return an immutable tuple object. The value at index number 1 is the number of days in the month.

```
year = 2020
month = 2
```

```
#get number of days in month
days_in_month = calendar.monthrange(year, month)[1] #index 1
```



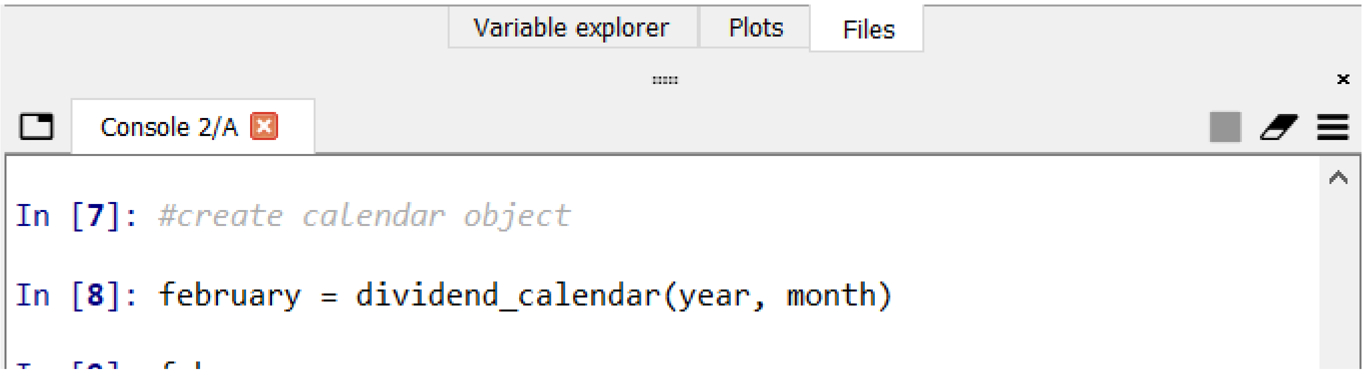
```
In [2]: year = 2020

In [3]: month = 2

In [4]: #get number of days in month

In [5]: days_in_month = calendar.monthrange(year, month)[1]

In [6]: days_in_month
Out[6]: 29

In [7]:
```

Monthrange method of calendar module at index 1 of the resulting tuple object.

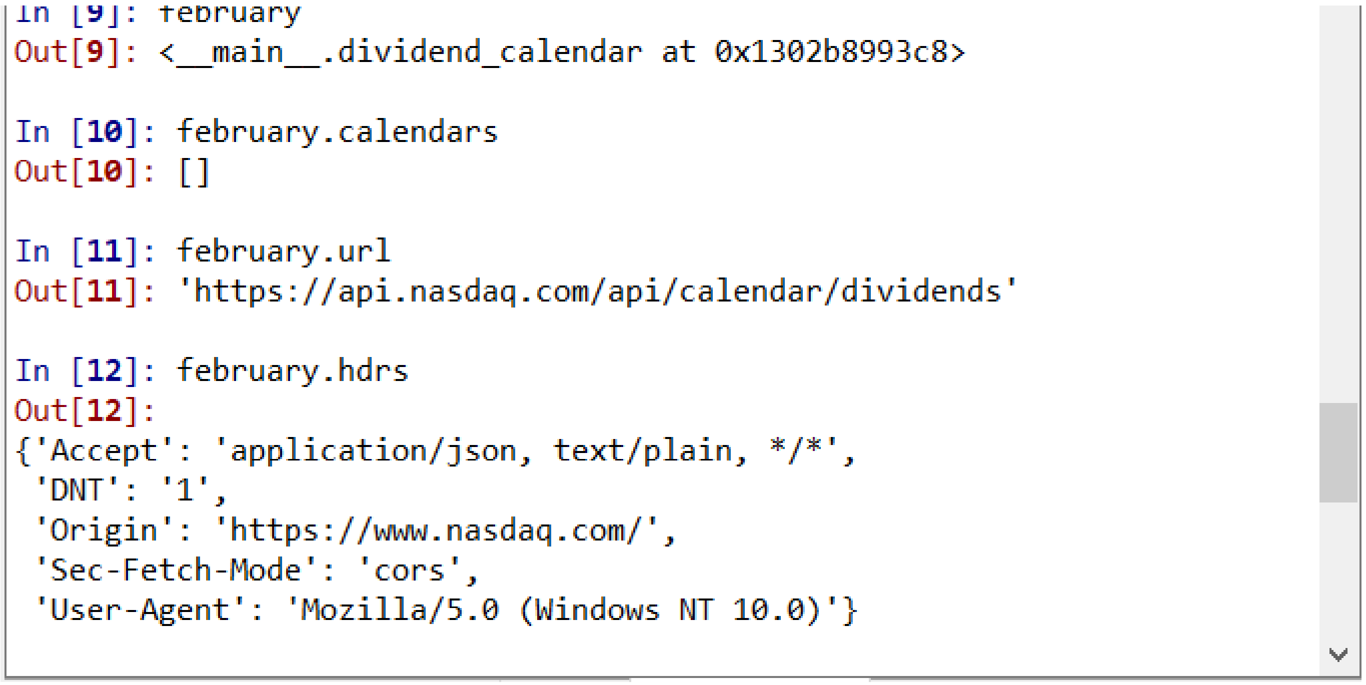
. . .

Step 2: *Create calendar object.* To create a calendar object, we simply call the class name and pass values in for year and month, the mandatory positional keyword arguments.

```
#create calendar object
february = dividend_calendar(year, month)
```



```
In [7]: #create calendar object

In [8]: february = dividend_calendar(year, month)
```



```
In [9]: february
Out[9]: <__main__.dividend_calendar at 0x1302b8993c8>

In [10]: february.calendars
Out[10]: []

In [11]: february.url
Out[11]: 'https://api.nasdaq.com/api/calendar/dividends'

In [12]: february.hdrs
Out[12]:
{'Accept': 'application/json, text/plain, */*',
 'DNT': '1',
 'Origin': 'https://www.nasdaq.com/',
 'Sec-Fetch-Mode': 'cors',
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0)'}
```

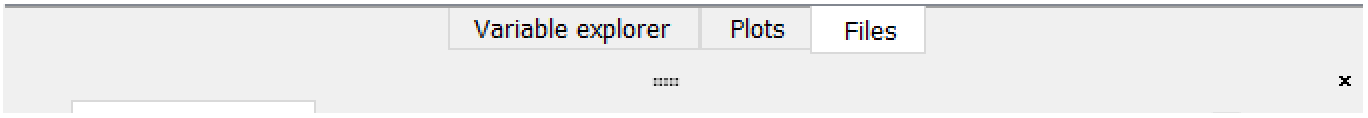
The dividend_calendar object and class attributes.

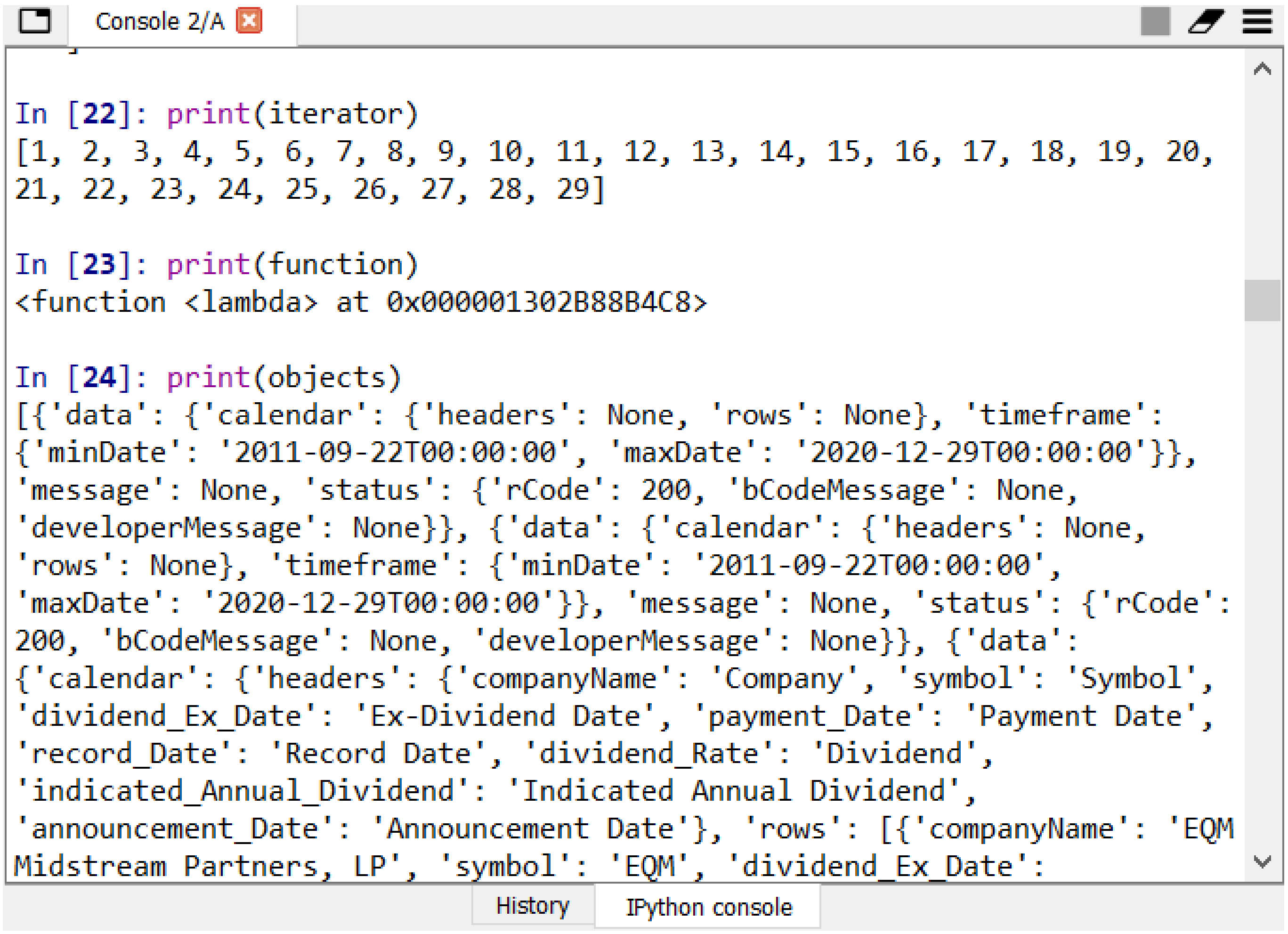
. . .

Step 3: *Scrape calendar for each day of the month.* The most efficient way to do this is by mapping a lambda function to a list iterator containing values representing all the days in the month.

```
#define lambda function to iterate over list of days
function = lambda days: february.calendar(days)

#define list of ints between 1 and the number of days in the month
iterator = list(range(1, days_in_month+1))

#Scrape calendar for each day of the month
objects = list(map(function, iterator))
```

> *Quick note:* Our iterator is in the range up until the last day of the month plus one because the list will not include the last number in the range. So, if there are 29 days in February, we must create a list in a range of maximum 30 days or `days_in_month+1` .





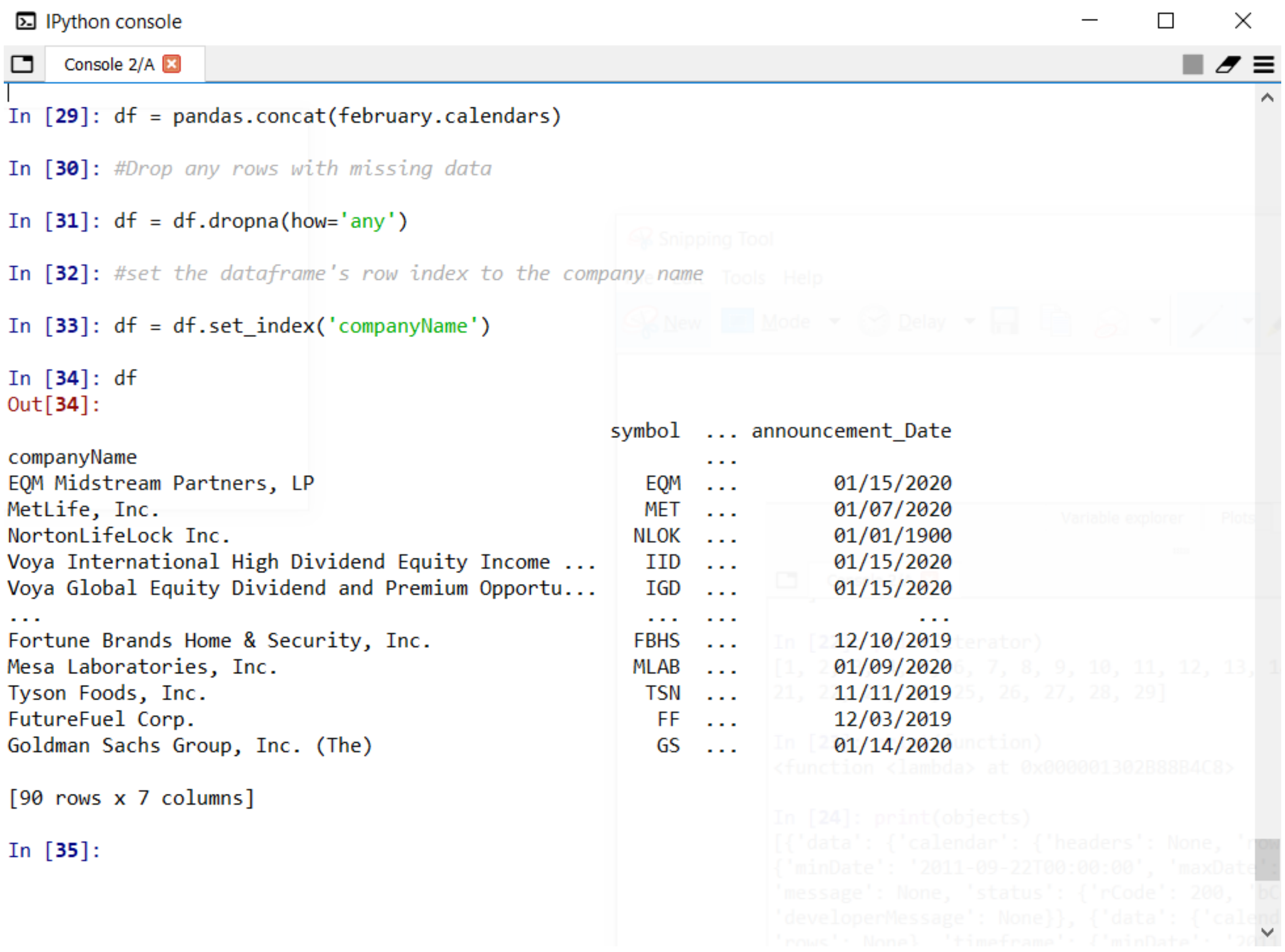
```
In [22]: print(iterator)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29]

In [23]: print(function)
<function <lambda> at 0x000001302B88B4C8>

In [24]: print(objects)
[{'data': {'calendar': {'headers': None, 'rows': None}, 'timeframe':
{'minDate': '2011-09-22T00:00:00', 'maxDate': '2020-12-29T00:00:00'}},
'message': None, 'status': {'rCode': 200, 'bCodeMessage': None,
'developerMessage': None}}, {'data': {'calendar': {'headers': None,
'rows': None}, 'timeframe': {'minDate': '2011-09-22T00:00:00',
'maxDate': '2020-12-29T00:00:00'}}, 'message': None, 'status': {'rCode':
200, 'bCodeMessage': None, 'developerMessage': None}}, {'data':
{'calendar': {'headers': {'companyName': 'Company', 'symbol': 'Symbol',
'dividend_Ex_Date': 'Ex-Dividend Date', 'payment_Date': 'Payment Date',
'record_Date': 'Record Date', 'dividend_Rate': 'Dividend',
'indicated_Annual_Dividend': 'Indicated Annual Dividend',
'announcement_Date': 'Announcement Date'}, 'rows': [{'companyName': 'EQM
Midstream Partners, LP', 'symbol': 'EQM', 'dividend_Ex_Date':
```

Lambda function, iterator containing all days in a month, and list of lambda mapped function output.

. . .

Step 4: *Concatenate class instances into one pandas dataframe.* We call the **calendars attribute** on `february` to retrieve a list of all the dataframes that we created in our current Python kernel. This makes it easy for us to then use the `pandas.concat` method to combine all the daily dataframes into one master dataframe for the month of February.

```
#concatenate all the calendars in the class attribute
concat_df = pandas.concat(february.calendars)

#Drop any rows with missing data
drop_df = concat_df.dropna(how='any')

#set the dataframe's row index to the company name
final_df = drop_df.set_index('companyName')
```



```
In [29]: df = pandas.concat(february.calendars)

In [30]: #Drop any rows with missing data

In [31]: df = df.dropna(how='any')

In [32]: #set the dataframe's row index to the company name

In [33]: df = df.set_index('companyName')

In [34]: df
Out[34]:
                                                  symbol  ... announcement_Date
companyName                                                ...
EQM Midstream Partners, LP                           EQM  ...        01/15/2020
MetLife, Inc.                                        MET  ...        01/07/2020
NortonLifeLock Inc.                                 NLOK  ...        01/01/1900
Voya International High Dividend Equity Income ...   IID  ...        01/15/2020
Voya Global Equity Dividend and Premium Opportu...   IGD  ...        01/15/2020
...                                                  ...  ...               ...
Fortune Brands Home & Security, Inc.                FBHS  ...        12/10/2019
Mesa Laboratories, Inc.                             MLAB  ...        01/09/2020
Tyson Foods, Inc.                                    TSN  ...        11/11/2019
FutureFuel Corp.                                      FF  ...        12/03/2019
Goldman Sachs Group, Inc. (The)                       GS  ...        01/14/2020

[90 rows x 7 columns]

In [35]:
```

Concat class attribute of the Python list of pandas dataframes, drop rows containing NA, and set index.

. . .

Putting it all together: Your final Python program

```
import pandas, requests, datetime, calendar

class dividend_calendar:
    #class attributes
    calendars = []
    url = 'https://api.nasdaq.com/api/calendar/dividends'
    hdrs =  {'Accept': 'application/json, text/plain, */*',
                 'DNT': "1",
                 'Origin': 'https://www.nasdaq.com/',
                 'Sec-Fetch-Mode': 'cors',
                 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0)'}
```

```
def __init__(self, year, month):
          '''
        Parameters
        ----------
        year : year int
        month : month int

        Returns
        -------
        Sets instance attributes for year and month of object.

          '''
          #instance attributes
          self.year = int(year)
          self.month = int(month)

     def date_str(self, day):
          date_obj = datetime.date(self.year, self.month, day)
          date_str = date_obj.strftime(format='%Y-%m-%d')
          return date_str

     def scraper(self, date_str):
         '''
          Scrapes JSON object from page using requests module.

           Parameters
           - - - - -
           url : URL string
           hdrs : Header information
           date_str: string in yyyy-mm-dd format

           Returns
           - - - -
           dictionary : Returns a JSON dictionary at a given URL.

         '''
         params = {'date': date_str}
         page=requests.get(self.url,headers=self.hdrs,params=params)
         dictionary = page.json()
         return dictionary

     def dict_to_df(self, dicti):
         '''
         Converts the JSON dictionary into a pandas dataframe
         Appends the dataframe to calendars class attribute

         Parameters
         ----------
         dicti : Output from the scraper method as input.

         Returns
         -------
         calendar : Dataframe of stocks with that exdividend date
         Appends the dataframe to calendars class attribute

         If the date is formatted correctly, it will append a
         dataframe to the calendars list (class attribute).
         Otherwise, it will return an empty dataframe.
         '''

         rows = dicti.get('data').get('calendar').get('rows')
         calendar = pandas.DataFrame(rows)
         self.calendars.append(calendar)
         return calendar


    def calendar(self, day):
          '''
          Combines the scrape and dict_to_df methods

          Parameters
          ----------
          day : day of the month as string or number.

          Returns
          -------
          dictionary : Returns a JSON dictionary with keys
          dictionary.keys() => data, message, status

          Next Levels:
          dictionary['data'].keys() => calendar, timeframe
          dictionary['data']['calendar'].keys() => headers, rows
          dictionary['data']['calendar']['headers'] => column names
          dictionary['data']['calendar']['rows'] => dictionary list

          '''
          day = int(day)
          date_str = self.date_str(day)
          dictionary = self.scraper(date_str)
          self.dict_to_df(dictionary)
          return dictionary

if __name__ == '__main__':
    year = 2020
    month = 2

#get number of days in month
    days_in_month = calendar.monthrange(year, month)[1]

#create calendar object
    february = dividend_calendar(year, month)

#define lambda function to iterate over list of days
    function = lambda days: february.calendar(days)

#define list of ints between 1 and the number of days in the month
    iterator = list(range(1, days_in_month+1))
```

```
#Scrape calendar for each day of the month
    objects = list(map(function, iterator))

#concatenate all the calendars in the class attribute
    concat_df = pandas.concat(february.calendars)

#Drop any rows with missing data
    drop_df = concat_df.dropna(how='any')

#set the dataframe's row index to the company name
    final_df = drop_df.set_index('companyName')
```

Python Web Scraping Programming Finance Data Science