```
#Here is an example implementation of a Python class named ScriptData that fetches US stock data using Alpha Vantage:

import requests

class ScriptData:
    def __init__(self, api_key):
        self.api_key = api_key

    def fetch_stock_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&symbol={symbol}&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch stock data")

        stock_data = response.json()["Time Series (Daily)"]
        return stock_data
#This class takes an Alpha Vantage API key as input in its constructor and provides a method named fetch_stock_data which takes
#a stock symbol as input and returns the daily adjusted stock data for that symbol as a dictionary. The requests library is used
#to make the API call to Alpha Vantage and the json library is used to parse the response.

#updated implementation of the ScriptData class with the fetch_intraday_data method added:
import requests

class ScriptData:
    def __init__(self, api_key):
        self.api_key = api_key

    def fetch_stock_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&symbol={symbol}&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch stock data")

        stock_data = response.json()["Time Series (Daily)"]
        return stock_data

    def fetch_intraday_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol={symbol}&interval=1min&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch intraday data")

        intraday_data = response.json()["Time Series (1min)"]
        return intraday_data
#This implementation adds a new method named fetch_intraday_data which takes a stock symbol as input and returns the intraday stock data for that symbol
```

```
#as a dictionary. The requests library is used to make the API call to Alpha Vantage and the json library is used to parse the response.
#In this case, we are fetching data for a 1-minute interval

#implementation of the ScriptData class with the convert_intraday_data method added:
import pandas as pd
import requests

class ScriptData:
    def __init__(self, api_key):
        self.api_key = api_key

    def fetch_stock_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&symbol={symbol}&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch stock data")

        stock_data = response.json()["Time Series (Daily)"]
        return stock_data

    def fetch_intraday_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol={symbol}&interval=1min&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch intraday data")

        intraday_data = response.json()["Time Series (1min)"]
        return intraday_data

    def convert_intraday_data(self, symbol):
        intraday_data = self.fetch_intraday_data(symbol)
        df = pd.DataFrame.from_dict(intraday_data, orient="index")
        df = df.astype(float)
        df.index = pd.to_datetime(df.index)
        df.columns = ["open", "high", "low", "close", "volume"]
        return df
#This implementation adds a new method named convert_intraday_data which takes a stock symbol as input and returns the intraday stock data for that symbol
#as a Pandas DataFrame. The fetch_intraday_data method is called to get the intraday data as a dictionary,
#which is then converted to a DataFrame using the pd.DataFrame.from_dict method.
#The index of the DataFrame is converted to pandas Timestamps using pd.to_datetime,
#and the column names are set to the specified values.
#Finally, the DataFrame is returned.

#implementation of the ScriptData class with methods for overloading the __getitem__, __setitem__, and __contains__ operations:
import pandas as pd
import requests
```

```
class ScriptData:
    def __init__(self, api_key):
        self.api_key = api_key

    def fetch_stock_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&symbol={symbol}&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch stock data")

        stock_data = response.json()["Time Series (Daily)"]
        return stock_data

    def fetch_intraday_data(self, symbol):
        url = f"https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol={symbol}&interval=1min&apikey={self.api_key}"
        response = requests.get(url)

        if response.status_code != 200:
            raise ValueError("Unable to fetch intraday data")

        intraday_data = response.json()["Time Series (1min)"]
        return intraday_data

    def convert_intraday_data(self, symbol):
        intraday_data = self.fetch_intraday_data(symbol)
        df = pd.DataFrame.from_dict(intraday_data, orient="index")
        df = df.astype(float)
        df.index = pd.to_datetime(df.index)
        df.columns = ["open", "high", "low", "close", "volume"]
        return df

    def __getitem__(self, key):
        return self.convert_intraday_data(key)

    def __setitem__(self, key, value):
        raise NotImplementedError("ScriptData does not support item assignment")

    def __contains__(self, key):
        try:
            self.fetch_intraday_data(key)
            return True
        except:
            return False

#This implementation overloads the __getitem__, __setitem__, and __contains__ operations for the ScriptData class.

#The __getitem__ method is defined to call the convert_intraday_data method when the class instance is indexed with a key, so that data can be retrieved for a stock symbol as a DataFrame like so: data["AAPL"].

#The __setitem__ method is defined to raise a NotImplementedError when an attempt is made to set an it
```

em using an index. This is because the data fetched from Alpha Vantage is read-only.

#The __contains__ method is defined to attempt to fetch intraday data for the given key, and return True if the fetch is successful, or False otherwise. This allows for checking whether a stock symbol has intraday data available using the in operator, like so: "AAPL" in data.

#mplementation of the indicator1 function that takes a pandas DataFrame df and an integer timeperiod as inputs, and returns a new pandas DataFrame with two columns:

```
import pandas as pd

def indicator1(df, timeperiod):
    ma = df["close"].rolling(window=timeperiod).mean()
    result = pd.DataFrame({"timestamp": df["timestamp"], "indicator": ma})
    return result
```

#The indicator1 function uses the rolling method of a pandas Series to calculate a moving average of the close column of the input DataFrame df. The window argument of the rolling method is set to the timeperiod input, which determines the number of elements to be included in the moving average calculation.

#The function then constructs a new DataFrame with two columns, "timestamp" and "indicator", using the pd.DataFrame constructor. The "timestamp" column is simply copied from the "timestamp" column of the input DataFrame. The "indicator" column is set to the moving average calculated earlier.

#The resulting DataFrame is then returned by the function.

#implementation of the Strategy class that can fetch intraday historical data, compute an indicator, and generate a pandas DataFrame with signals:

```
from script_data import ScriptData
from indicator_functions import indicator1
import pandas as pd

class Strategy:
    def __init__(self):
        pass

    def get_signals(self, script):
        # Fetch intraday data
        sd = ScriptData()
        df = sd.fetch_intraday_data(script)
        # Compute indicator data
        indicator_data = indicator1(df, timeperiod=5)
        # Generate signals
        signals = pd.DataFrame({"timestamp": df["timestamp"], "signal": "NO_SIGNAL"})
        for i in range(1, len(df)):
            if indicator_data.iloc[i] > df["close"].iloc[i] and indicator_data.iloc[i-1] <= df["close"].iloc[i-1]:
                signals.loc[i, "signal"] = "BUY"
            elif indicator_data.iloc[i] < df["close"].iloc[i] and indicator_data.iloc[i-1] >= df["close"].iloc[i-1]:
                signals.loc[i, "signal"] = "SELL"
        return signals
```

#The Strategy class has a single method called get_signals that takes a script argument, which is used to fetch intraday historical data using the ScriptData class. The indicator1 function is then used to compute the indicator data.

#The signals DataFrame is then generated by first creating a DataFrame with a "timestamp" column and a "signal" column initialized to "NO_SIGNAL" for every row. The signals are then computed by iterating through the DataFrame and checking whether the indicator data crosses the close data, and updating the "signal" column accordingly.

#Note that the above implementation uses a simple crossover strategy, where a BUY signal is generated when the indicator crosses above the close data, and a SELL signal is generated when the indicator crosses below the close data. There are many other more complex strategies that could be implemented, depending on the specific needs of the user.