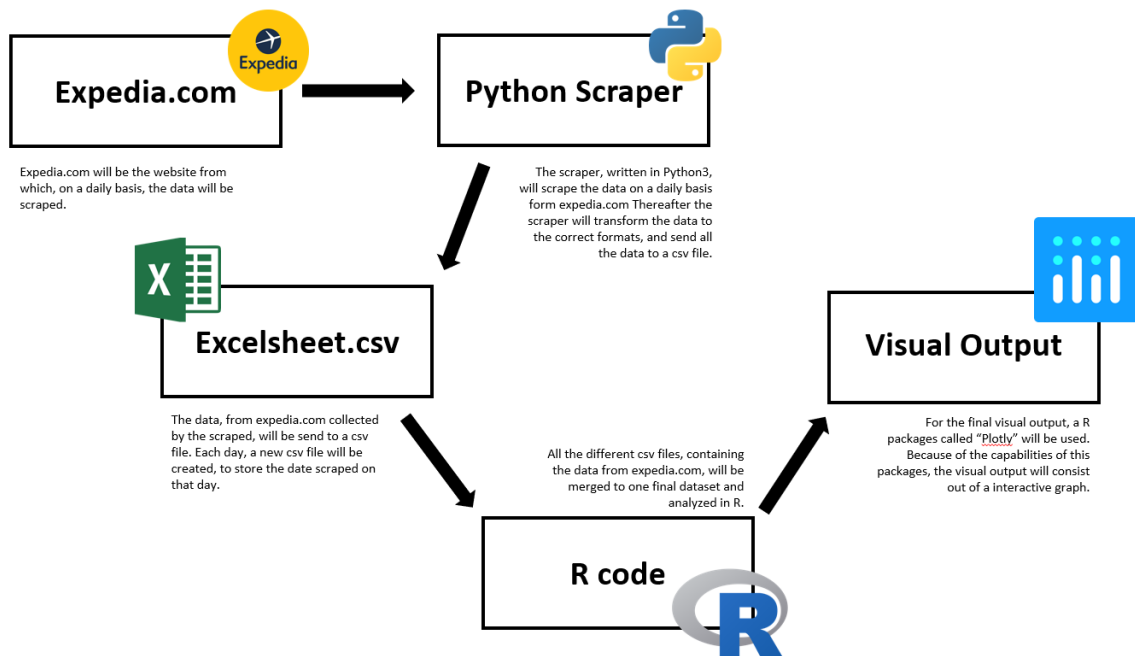


## User Story

Since I fly to LA quite frequently, in order to find the cheapest offer, I search the web on a regular basis. However, since prices of flight tickets change every day, this takes up quite some time. Also if I want to be certain of the cheapest offer, it means I'll have to manually execute the same search query every day. To solve this problem, I built my own flight ticket scraper in Python and visualized the data using R. This scrapers allows me to set the destination and the dates on which the flight must be taken place. Thereafter, the scraper will get its data from Expedia.com on a daily bases, which then will be visualized in R.

The final output will consist out of multiple graphs. First and foremost, there will be a graph which will show the price of the flight on the x-axis and the amount of days until the flight on the y-axis, per different airline. Also, there will be a more general graph which will show the average price of flight tickets per airline, trough time. These graphs will enable the user, to get an impression of the price-development of the flight tickets.

## Process flow



## Sources

### Source 1:

Category: Website

Url: Stackoverflow.com

#### *Why:*

Stackoverflow is one of the biggest website, where all developers come together and share solutions with one and other. Whenever you have a problem, regarding the code of your project/product, you can either use the platform to search for a similar question and answer, or ask the question yourself. In this way, Stackoverflow can help out 99% of the time.

### Source 2:

Category: Video/YouTube

Url: <https://www.youtube.com/watch?v=XQgXKtPSzUI>

#### *Why:*

This video is a clear and well structured introduction to web scraping and using Python as a programming language for building a web scraper. After watching this video, you'll have the most basic, but also required knowledge to start your own project.

### Source 3:

Category: Video/YouTube

Url: <https://www.youtube.com/user/hiteshitube>

#### *Why:*

Just like the other video, also this video provides some crucial information, regarding building your own web scraper. Also this video provides sources that will help you with building your own web scraper.

# Tutorial

## Essentials

Before looking at the coding itself, there are a couple of essential components that needs to be addressed. First of all, let's define the software which will be used during this tutorial.

### Software

#### *Python3*

The software which will used for building the web scraper, is Python3. In this case 3 stands for the version of python which will be used. When using Python2, the same result can be achieved, however this tutorial won't be the ideal instruction. To download Python3, follow this link: <https://www.python.org/download/releases/3.0/>

#### *R – Statistical Analysis (will not be included in the tutorial)*

As for the analysis and final output of this project, R will be the software which will be used. With R the data will be manipulated and finally put in one, interactive, graph. This graph will be created with a R-packages called: "Plotly".

#### *Excel*

Excel will be used as a middleman between Python and R. When the data is craped by the python script, it will be stored in a csv file. Then when analyzing the data, R will use this csv file and use to data form this csv file, to create the final output, the interactive graph.

## Web scraper (Script)

### Modules

Before we start writing the actual script, we first need to make sure we import a couple of modules. These modules contain functions which enables us to build the scraper, in the way we want.

```
import json
import requests
from lxml import html
from collections import OrderedDict
import datetime
import re
```

Figure 1

**Json:** The first module we see is Json, this packages is used to transform the scraped data from the website, in a desired format.

**Requests:** This module creates the connection with the website, from which the data will be scraped.

**Html:** The module html is used to read the html of a website and the Javascripts that are within this html.

**OrderedDict:** An OrderedDict is a dictionary subclass that remembers the order in which its contents are added.

**Datetime:** The datetime module is used for everything involving dates and time variables.

**Re:** The re module is used for everything involving regular expression

```
departure_date = "02%2F28%2F2019"
return_date = "03%2F14%2F2019"
url = "https://www.expedia.com/Flights-Search?flight-type=on&starDate="+departure_date+"&endDate="+return_date+"&mode=search&trip=roundtrip&le
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36'}
response = requests.get(url, headers=headers)
parser = html.fromstring(response.text)
json_data_xpath = parser.xpath("//script[@id='cachedResultsJson']//text()")
raw_json = json.loads(json_data_xpath[0] if json_data_xpath else '')
flight_data = json.loads(raw_json["content"])
```

Figure 2

The Code

### Connecting and getting the data

Above screenshot is from the part of the script where the connection with the website is being made. First there are two variables being defined, namely “departure\_date” and return\_date”. These variables contain the departure and return date of the flight, of which the prices will be tracked. These variables will be joined with “url” variable, by the “+” symbols you will find in the url variable. After the url follow a couple of variable that determine the initial input for the script. As you can see in the “raw\_json” variable, on the website of Expedia there is a script called “cachedResultsJson” (see figure3).

```
<div class="dCol">...</div>
<div style="display: none;" id="location-agnostic-includes">
  <script>...</script>
  <script id="cachedResultsJson" type="application/json" data-test-id=
    "cacheRendered">...</script>
  <script id="singlePageModel" type="application/json">...</script>
```

Figure 3

This script contains all the flight data (see figure 4), in a xml format. This data will be used for the input of our script. At the end of the script (see figure 1) all the data is stored in the variable “flight\_data”

```
{
  "content": {
    "legs": {
      "4f856ec63ce001122a8e0455cab7537e": {
        "identity": {
          "naturalKey": "4f856ec63ce001122a8e0455cab7537e",
          "index": 8,
          "fareBasisCode": "ALCRS26R",
          "index": 8,
          "naturalKey": "4f856ec63ce001122a8e0455cab7537e",
          "price": {
            "asFare": true,
            "flightFareTypeCode": "P",
            "flightFareTypeValue": 1,
            "priceIllegal": false,
            "offerPrice": 638.0,
            "exactPrice": 637.5,
            "formattedPrice": "$637.50",
            "formattedPriceWithCreditCardFeesEstimate": "",
            "formattedTotalPrice": "$637.50",
            "totalPriceAsDecimal": 637.5,
            "totalPriceAsString": "637.50",
            "formattedRoundedPrice": "$638",
            "formattedRoundedTotalPrice": "$638",
            "currencyCode": "USD",
            "feesMessage": {
              "isShowBestPriceGuarantee": false,
              "showBaggageFeeIncludedMsg": false,
              "showCheckedBagIncludedMsg": false,
              "showFirstCheckedBagIncludedMsg": false,
              "showLCCBaggageAllowanceMsg": false,
              "baggageAllowance": "",
              "showOBFeeMessageForLeg": false,
              "isShowFreeCancellation": true,
              "showBaggageFeeNotIncludedMsg": false,
              "showBaggageFeeOnPurchaseMsg": false,
              "showHandBaggageOnlyMsg": false,
              "airlineBasedBaggageAllowance": 0,
              "specialFareMessage": "",
              "freeCancellationTimeLimit": 0,
              "showOBFeeDetailLink": false,
              "nextGenContentWithLink": false,
              "matchDataAirProvider": true,
              "isFlightIneligibleForBPG": false,
              "localizedCurrencyCode": "USD",
              "bestPriceDelta": 167.97000000000003,
              "formattedBestPriceDelta": "$167.97",
              "formattedRoundedBestPriceDelta": "$168",
              "roundedBestPriceDelta": 168,
              "earnGPSREwards": null,
              "hasFees": false,
              "pricedFlight": false,
              "carrierSummary": {
                "airlineImageFileName": "EI.gif",
                "airProviderId": 2,
                "seatMapAvailable": true,
                "airlineCodes": [
                  "EI",
                  "EI"
                ],
                "mixedCabinClass": false,
                "daysTravelled": 0,
                "displayUrgencyMessage": false,
                "noOfTicketsLeft": 10,
                "airlineName": "Aer Lingus",
                "multistop": true,
                "nextDayArrival": true,
                "timeline": [
                  {
                    "type": "Segment",
                    "segment": true,
                    "layover": false,
                    "carrier": {
                      "flightNumber": 611,
                      "seatMapAvailable": true,
                      "airlineName": "Aer Lingus",
                      "airlineImageFileName": "EI.gif",
                      "airlineCode": "EI",
                      "cabinClass": "3",
                      "bookingCode": "Z",
                      "operatedBy": "",
                      "operate": "Airbus A320",
                      "planeCode": "320",
                      "airlineImage": "Airbus A320",
                      "showCabinClass": true,
                      "departureAirport": "..."
                    }
                  }
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 4

```
filename = "Flights " + "-" + str(datetime.datetime.now().date()) + ".csv"
f = open(filename, "w")

headers = "Price, Departure Location Airport, Arrival Location Airport, Airline, Plane, Stops, Plane Code, Departure Time, Arrival Time, Date of Scrape, Days until departure\n"
f.write(headers)
```

Figure 5

### Creating a CSV document

On the screenshot above, the csv file in which the data will be stored, is being created. First of all, the name of the csv file is created by the variable "filename". This variable creates a filename using the date of when the scrape is executed. Nextup, the headers are created by the variable called "Headers". Using the function(f.write(headers)) the headers get written to the csv file.

```
for i in flight_data['legs'].keys():
    exact_price = flight_data['legs'][i].get('price', {}).get('totalPriceAsDecimal', '')

    departure_location_airport = flight_data['legs'][i].get('departureLocation', {}).get('airportLongName', '')
    departure_location_city = flight_data['legs'][i].get('departureLocation', {}).get('airportCity', '')

    arrival_location_airport = flight_data['legs'][i].get('arrivalLocation', {}).get('airportLongName', '')
    arrival_location_city = flight_data['legs'][i].get('arrivalLocation', {}).get('airportCity', '')

    airline_name = flight_data['legs'][i].get('carrierSummary', {}).get('airlineName', '')
    carrier = flight_data['legs'][i].get('timeline', [])[0].get('carrier', {})

    if not airline_name:
        airline_name = "Undefined"

    departure = departure_location_airport + ", " + departure_location_city
    arrival = arrival_location_airport + ", " + arrival_location_city
```

Figure 6

### Getting flight ticket information

The code on the image above, gets for every different ticket the information regarding the price, departure location, arrival location, airline name and the carrier, using a for-loop.

First of all you will find the for-loop, which loops through all the different flights that are saved in the variable called "flight\_data". It loops through all the "legs", which resembles the different flights. For each leg it gets the following information: exact\_price, departure\_location\_airport, departure\_location\_city, arrival\_location\_city, arrival\_location\_city, airline\_name, carrier. Here is how these variables are being extracted from the big pile of information:

**Step 1:** First the variable which contains all the data get appointed (in this case flight\_data).

```
{"content": "{ \"legs\": { \"4f856ec63ce001122a8e0455cab7537e\":
```

Figure 7

**Step 2:** Within the dataset, the string 'legs' get called, in which all the different flight data is stored (see figure 7).

**Step 3:** The [i] resembles the index of the particular value in the code, which is being extracted. By using this [i], Python knows which index to look for, since the same variables appear multiple times in the code, due to the different flight ticket information that is stored in the code.

**Step 4:** After the [i], follows the “.get” function. By calling this function, Python knows that it should get and return something from the code which will be within the “.get()” function.

**Step 5:** Since the first variable to extract from the code is the price, the variable within “.get()” is price.

**Step 6:** However, since the variable “price” itself hasn’t any direct value, though it contains several sub variables, “{” is added. This means that Python will look within the array of “price” with the next “.get()” function.

**Step 7:** To get the actual price as decimals from every ticket, .get(“totalPriceAsDecimal”, “”) is being added. This piece of code looks within the array “price” for a variable called “totalPriceAsDecimal”. (see figure 8)

Figure 8

**Step 8:** Repeat the same steps for every variable you want to extract from the data, in order to get the data that is needed.

```
for timeline in flight_data['legs'][i].get('timeline', {}):
    if 'departureAirport' in timeline.keys():
        rep = {"uuu": "", ".": ":"}
        rep = dict((re.escape(k), v) for k, v in rep.items())
        pattern = re.compile("|".join(rep.keys()))

        departure_time = timeline['departureTime'].get('time', '')
        clean_departure_time = pattern.sub(Lambda m: rep[re.escape(m.group(0))], departure_time)
        arrival_time = timeline.get('arrivalTime', {}).get('time', '')
        clean_arrival_time = pattern.sub(Lambda m: rep[re.escape(m.group(0))], arrival_time)

        clean_departure_time = str(clean_departure_time)
        clean_arrival_time = str(clean_arrival_time)
```

Figure 9

The same steps as explained above, go for extracting data regarding the time variables of the different flights. However since all the data regarding time is stored in the array called “Timeline”, we use a for-loop within the initial for-loop. This keeps the code much simpler and clearer. (See figure 9)

## Changing the data

As can be seen on figure 10, the output of the variables, which we extracted from the data during the previous steps, is adjusted in order to fit our requirements. For example, when some of the data is not available, since sometimes Expedia.com

```
plane = carrier.get('plane','')
if not plane:
    plane = "Undefined"
plane_code = carrier.get('planeCode','')
if not plane_code:
    plane_code = "Undefined"
price = "{0:.2f}".format(exact_price)
if not price:
    price = "Undefined"

no_of_stops = flight_data['legs'][i].get("stops","")

if no_of_stops==0:
    stop = "Nonstop"
elif no_of_stops==1:
    stop = str(no_of_stops)+' stop'
else:
```

Figure 10

does not provide all the required data, we want it to return "Undefined" instead of just an NA value. Also in this case the output of the amount of stops, is adjusted depending on the amount of stops. All can be done to, in the end, get the desired output. Keep in mind that this is just some extra finetuning, instead a requirement.

```
price = str(price)
departure_location_city = str(departure_location_city)
airline_name = str(airline_name)
plane = str(plane)
stop = str(stop)
plane_code = str(plane_code)
```

Figure 11

## Converting the variables to strings

As can be seen on figure 11, all the variables which were assigned earlier on in the code, are converted to the variable type called "string". This means they are converted to just text, which is necessary in order to create the final output. When skipping this part, Python will give an error, when writing the data to the final document, since it can only write strings to a csv file, and for example not a datetime or Boolean variable.

```
departure_date_clean = re.sub("-", " ", departure_date)
datetime_object = datetime.datetime.strptime(departure_date_clean, '%m %d %Y')

date_of_scrape = datetime.datetime.now().strftime('%m %d %Y')
datescrape_object = datetime.datetime.strptime(date_of_scrape, '%m %d %Y')

time_until_departure = datetime_object - datescrape_object
time_until_departure_object = str(time_until_departure)
time_until_departure_object_clean = re.sub("days, 0:00:00", "", time_until_departure_object)
```

Figure 12

## Calculating time between day of scrape and day of departure

The code in figure 12, shows a calculation of the amount of days that are in between the day on which the data is being scraped, and the day of departure. In this case the data is scraped on 14-01-2019 and the date of departure is on 28-02-2019. This means that there are 45 days in between. This variable will be used when visualizing the data, to get an impression of how the price evolves when getting closer to the date of departure.

```

date_dis = str(datetime.datetime.now().date())
f.write(price + "," + departure_location_city + "," + arrival_location_city + "," + airline_name + "," + plane + "," + stop + "," + plane_code + "," +
        + clean_departure_time + "," + clean_arrival_time + "," + date_dis + "," + time_until_departure_object_clean + "\n")
f.close()

```

Figure 13

### Writing the data to a csv file

Figure 13 contains the last part of the code. This piece of code makes sure that all the data, which is extracted and processed by all the previous lines of code, gets written to a csv file. The function “f.write()”, writes all the variables, which are included within this function, to the csv file which has earlier in the code been created. The “f.close()” function closes the entire process.

### Executing the code

In order to execute the Python scraper, we need to make use of cmd. Cmd is the command prompt of any windows laptop or pc. To start with, we first need to save the script to a file location. In this case the location of the script is:

- Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data

In order to access this folder using cmd, we use the “cd” function. This allows us to change the directory within cmd, so we can execute the python script. (see figure 14)

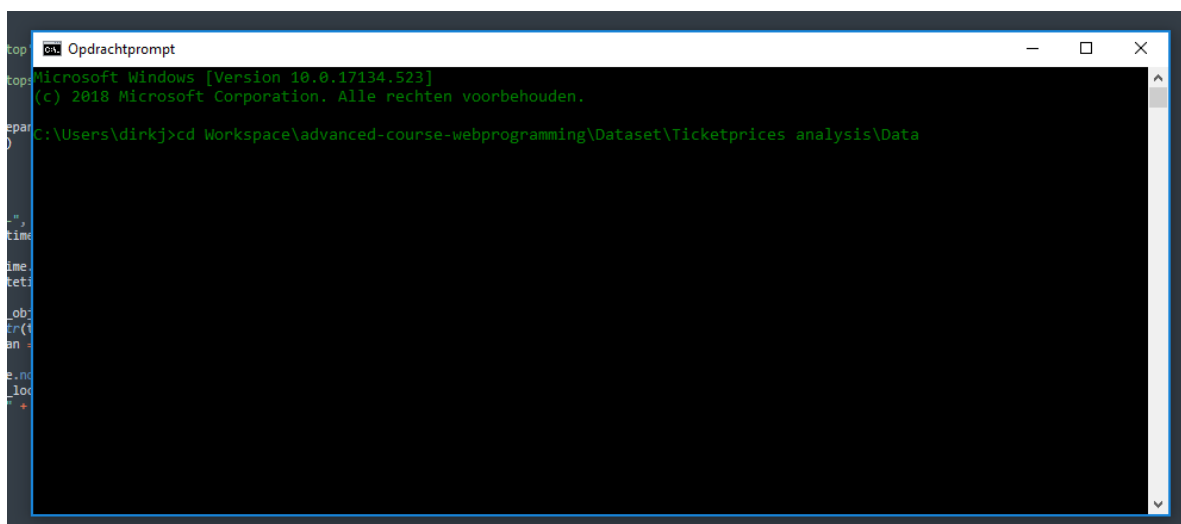
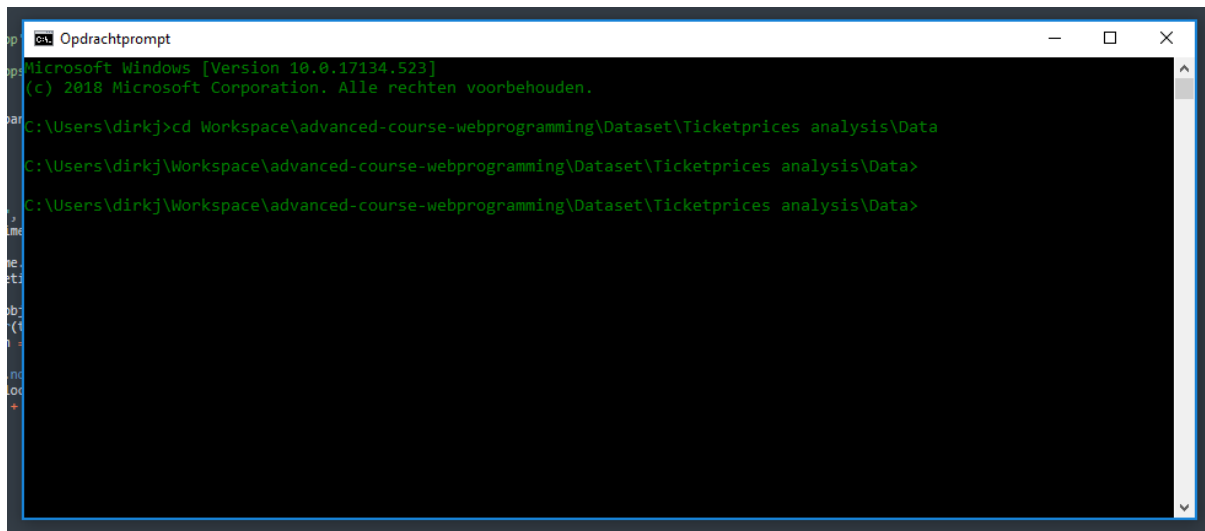


Figure 14

When hitting “Enter” the directory in cmd will change from “C:\Users\dirkj>”, to “C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>” (See figure 15)

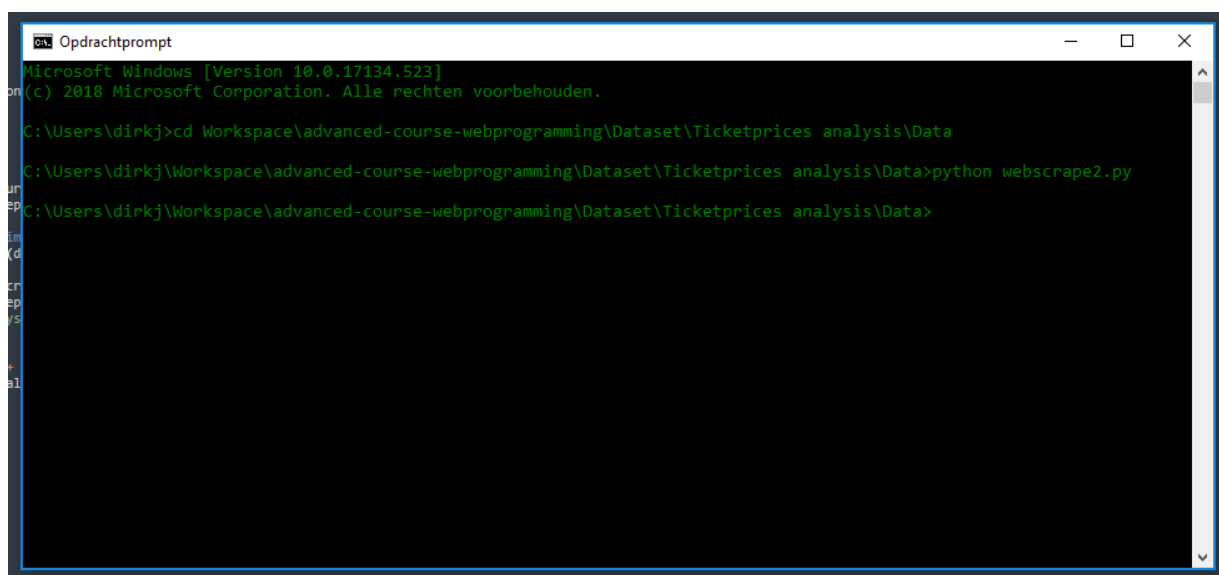




```
Opdrachtprompt
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\dirkj>cd Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>
```

Figure 16



```
Opdrachtprompt
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\dirkj>cd Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>python webscrape2.py
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>
```

Figure 15

When you're in the subdirectory, which contains the code that needs to be executed, type the following: `python filename.py`. This will execute the code (see figure 16).

Now the code is executed, we navigated to the directory in which the code is executed. If everything went well and no errors appeared, there will be a csv file containing all the data that was scraped. (see figure 17)

As can be seen on figure 17, the code was well executed and the csv-file named "Flights\_2019-01-14" was created. If we open this file we will see the following data. (see figure 20)

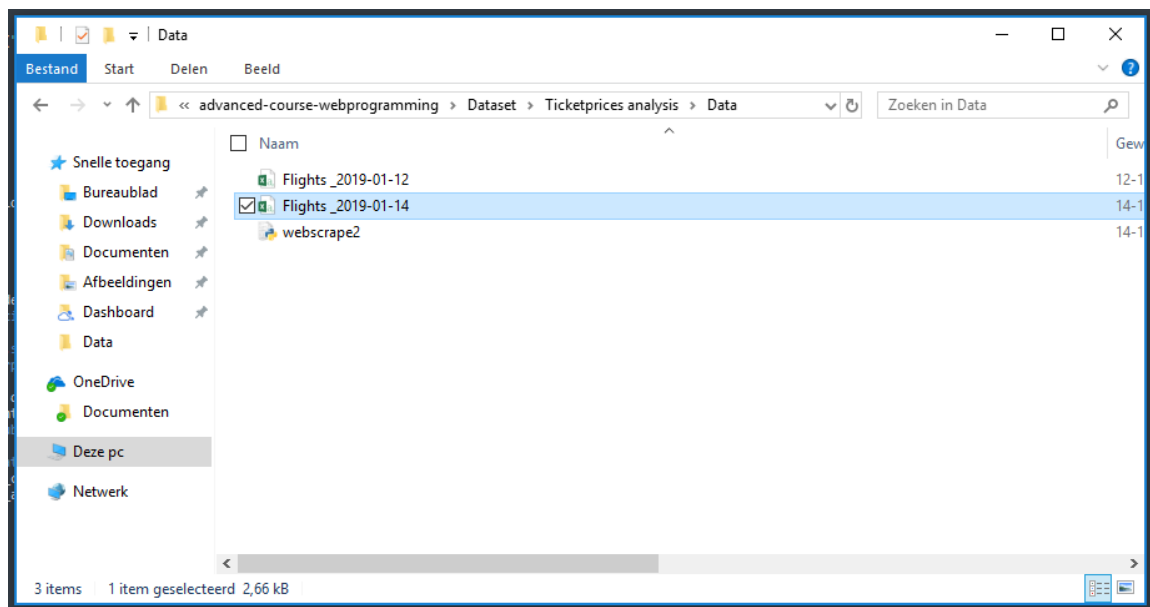


Figure 18

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	Price	Departure Location	Airport	Arrival Location	Airport	Airline	Plane	Stops	Plane Code	Departure Time	Arrival Time	Date of Scrape	Days until departure		
1	603.50	Amsterdam	Los Angeles	Aer Lingus	Airbus A320	1 Stop	320	3:15pm	6:20pm	2019-01-14	45				
2	509.83	Los Angeles	Amsterdam	Swiss International Air Lines	BOEING 777-300ER	1 Stop	77W	5:35pm	7:20pm	2019-01-14	45				
3	469.53	Amsterdam	Los Angeles	British Airways	Airbus A320	1 Stop	320	12:30pm	3:55pm	2019-01-14	45				
4	518.93	Amsterdam	Los Angeles	United	Boeing 767-300	1 Stop	763	3:50pm	6:15pm	2019-01-14	45				
5	519.43	Amsterdam	Los Angeles	Lufthansa	Airbus A320	1 Stop	320	11:50am	3:25pm	2019-01-14	45				
6	602.43	Amsterdam	Los Angeles	Delta	AIRBUS INDUSTRIE A330-300	Nonstop	333	3:30pm	6:17pm	2019-01-14	45				
7	602.43	Los Angeles	Amsterdam	Delta	Boeing 747-400	Nonstop	744	2:50pm	9:05am	2019-01-14	45				
8	603.50	Los Angeles	Amsterdam	Aer Lingus	AIRBUS INDUSTRIE A330-200	1 Stop	332	5:00pm	7:45pm	2019-01-14	45				
9	612.53	Amsterdam	Los Angeles	Delta	AIRBUS INDUSTRIE A330-300	1 Stop	333	8:25pm	10:26pm	2019-01-14	45				
10	469.53	Amsterdam	Los Angeles	Undefined	Airbus A320	1 Stop	320	12:30pm	3:55pm	2019-01-14	45				
11	615.13	Amsterdam	Los Angeles	KLM	Boeing 747-400	Nonstop	74E	9:55am	11:55am	2019-01-14	45				
12	469.53	Amsterdam	Los Angeles	British Airways	Airbus A321	1 Stop	321	3:30pm	6:50pm	2019-01-14	45				
13	602.43	Amsterdam	Los Angeles	Delta	Boeing 747-400	Nonstop	74E	9:55am	11:55am	2019-01-14	45				
14	612.53	Amsterdam	Los Angeles	Delta	AIRBUS INDUSTRIE A330-300	1 Stop	333	2:38pm	4:40pm	2019-01-14	45				
15	509.83	Amsterdam	Los Angeles	United	AIRBUS INDUSTRIE A320 SHARKLETS	1 Stop	32A	10:25am	1:20pm	2019-01-14	45				
16	512.22	Amsterdam	Los Angeles	Air Canada	AIRBUS INDUSTRIE A330-300	1 Stop	333	4:20pm	6:53pm	2019-01-14	45				
17	469.53	Amsterdam	Los Angeles	American Airlines	Airbus A320	1 Stop	320	12:30pm	3:55pm	2019-01-14	45				
18	513.33	Los Angeles	Amsterdam	United	Boeing 757 (757-300)	1 Stop	753	5:55pm	8:15am	2019-01-14	45				
19	518.13	Amsterdam	Los Angeles	Lufthansa	AIRBUS INDUSTRIE A320 SHARKLETS	1 Stop	32A	10:25am	1:20pm	2019-01-14	45				
20	510.53	Amsterdam	Los Angeles	British Airways	Airbus A321	1 Stop	321	10:35am	1:55pm	2019-01-14	45				
21	620.23	Amsterdam	Los Angeles	Air France	BOEING 737-800 (WINGLETS) PASSENGER	1 Stop	73H	10:20am	1:05pm	2019-01-14	45				
22	469.53	Los Angeles	Amsterdam	British Airways	Airbus Industrie A380-800 Passenger	1 Stop	388	4:25pm	6:45pm	2019-01-14	45				
23	513.33	Amsterdam	Los Angeles	Swiss International Air Lines	Undefined	1 Stop	Undefined	1:10pm	4:20pm	2019-01-14	45				
24	615.13	Amsterdam	Los Angeles	KLM	AIRBUS INDUSTRIE A330-300	Nonstop	333	3:30pm	6:17pm	2019-01-14	45				
25	469.53	Amsterdam	Los Angeles	American Airlines	Airbus A320	1 Stop	320	1:10pm	4:40pm	2019-01-14	45				

Figure 17

The data on figure 20 is in a csv format, meaning all variables are separated by a comma. In to get a clearer overview of our final output, we need to split the data to columns, using the build-in function of Excel. (see figure 18)

When selecting this build-in function, you'll be able to use a couple of different methods to separate the data. In our case, since the data is separated by commas, we need to select the comma as being the separator. (See figure 19)

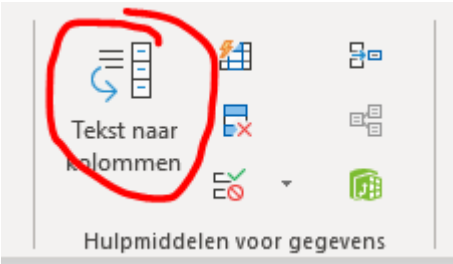


Figure 19

After selecting “comma” as a separator, click on the button “complete”. Your data will be separated and, if done right, your data will look like data in figure 21.

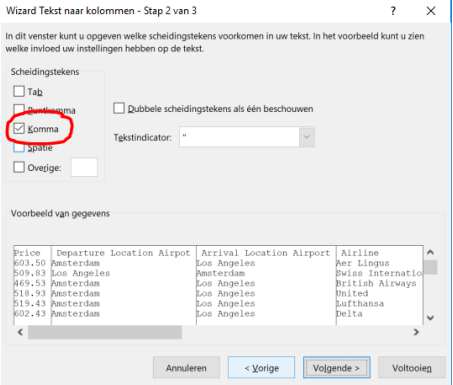


Figure 20

After selecting “comma” as a separator, click on the button “complete”. Your data will be separated and, if done right, your data will look like data in figure 21.

Price	Departure Location Airport	Arrival Location Airport	Airline	Plane	Stops	Plane Code	Departure Time	Arrival Time	Date of forage	Days until departure
603.50	Amsterdam	Los Angeles	Aer Lingus	Airbus A320	1 Stop	320	11:15pm	6:20pm	15-1-2019	44
509.83	Los Angeles	Amsterdam	Swiss International Air Lines	BOEING 777-300ER	1 Stop	77W	5:35pm	7:20pm	15-1-2019	44
469.53	Amsterdam	Los Angeles	British Airways	Airbus A320	1 Stop	320	12:30pm	1:55pm	15-1-2019	44
519.93	Amsterdam	Los Angeles	United	Boeing 767-300	1 Stop	763	3:50pm	6:15pm	15-1-2019	44
519.43	Amsterdam	Los Angeles	Lufthansa	Airbus A320	1 Stop	320	11:50am	3:25pm	15-1-2019	44
602.43	Amsterdam	Los Angeles	Delta	AIRBUS INDUSTRIE A330-300	Nonstop	333	3:30pm	6:17pm	15-1-2019	44
602.43	Los Angeles	Amsterdam	Delta	Boeing 747-400	Nonstop	744	2:50pm	9:05am	15-1-2019	44
603.50	Los Angeles	Amsterdam	Aer Lingus	AIRBUS INDUSTRIE A330-300	1 Stop	332	5:00pm	7:45pm	15-1-2019	44
612.53	Amsterdam	Los Angeles	Delta	AIRBUS INDUSTRIE A330-300	1 Stop	333	8:25pm	10:26pm	15-1-2019	44
469.53	Amsterdam	Los Angeles	Undefined	Airbus A320	1 Stop	320	12:30pm	1:55pm	15-1-2019	44
615.13	Amsterdam	Los Angeles	KLM	Boeing 747-400	Nonstop	74E	9:55am	11:55am	15-1-2019	44
469.53	Amsterdam	Los Angeles	British Airways	Airbus A321	1 Stop	321	3:30pm	6:50pm	15-1-2019	44
602.43	Amsterdam	Los Angeles	Delta	Boeing 747-400	Nonstop	74E	9:55am	11:55am	15-1-2019	44
612.53	Amsterdam	Los Angeles	Delta	AIRBUS INDUSTRIE A330-300	1 Stop	332	3:28pm	4:40pm	15-1-2019	44
509.83	Amsterdam	Los Angeles	United	AIRBUS INDUSTRIE A320 SHARKLETS	1 Stop	32A	10:25am	1:20pm	15-1-2019	44
512.22	Amsterdam	Los Angeles	Air Canada	AIRBUS INDUSTRIE A330-300	1 Stop	333	4:20pm	6:53pm	15-1-2019	44
469.53	Amsterdam	Los Angeles	American Airlines	Airbus A320	1 Stop	320	12:30pm	1:55pm	15-1-2019	44
513.33	Los Angeles	Amsterdam	United	Boeing 757 (757-300)	1 Stop	753	5:55pm	8:15am	15-1-2019	44
518.13	Amsterdam	Los Angeles	Lufthansa	AIRBUS INDUSTRIE A320 SHARKLETS	1 Stop	32A	10:25am	1:20pm	15-1-2019	44
510.53	Amsterdam	Los Angeles	British Airways	Airbus A321	1 Stop	321	10:15am	1:55pm	15-1-2019	44
620.23	Amsterdam	Los Angeles	Air France	BOEING 737-800 (WINGLETS) PASSENGER	1 Stop	73H	10:20am	1:05pm	15-1-2019	44
469.53	Los Angeles	Amsterdam	British Airways	Airbus industrie A380-800 Passenger	1 Stop	388	4:25pm	6:45pm	15-1-2019	44
513.33	Amsterdam	Los Angeles	Swiss International Air Lines	Undefined	1 Stop	Undefined	1:10pm	4:20pm	15-1-2019	44
615.13	Amsterdam	Los Angeles	KLM	AIRBUS INDUSTRIE A330-300	Nonstop	333	3:30pm	6:17pm	15-1-2019	44
469.53	Amsterdam	Los Angeles	American Airlines	Airbus A320	1 Stop	320	1:10pm	4:40pm	15-1-2019	44

Figure 21

## Automation

Since we want to check the flight ticket prices every day using our Python script, this would mean we would have to execute it manually every day. However, there is a program which can automate this process, which is called “Taskmanager”. This program comes on every Windows PC and enables us to automatically run the script every day.

**Step 1:** To begin with, we need to create a task, which will run our script. As you can see on figure 22, this can be done by clicking on “create task”.

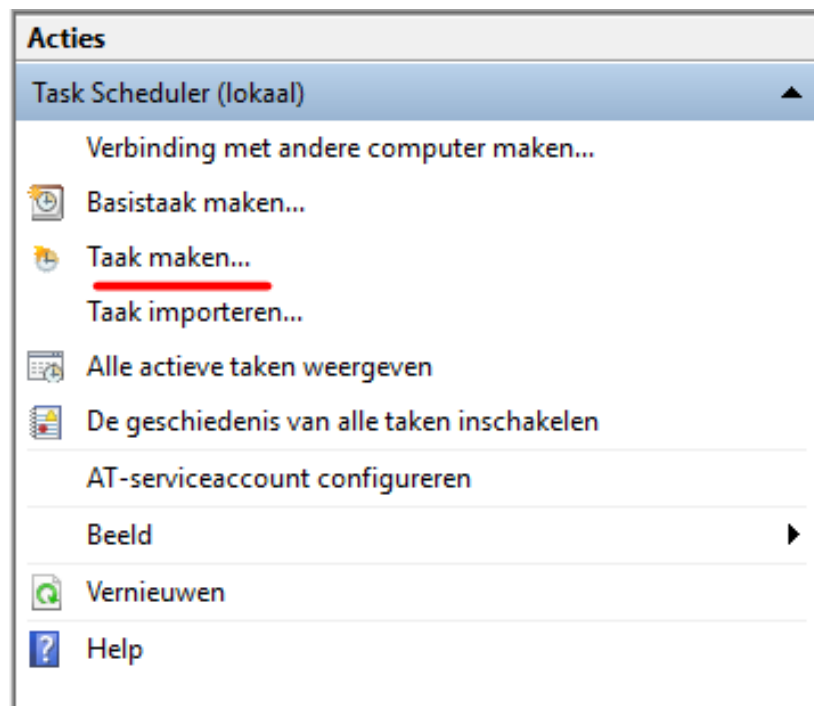


Figure 22

**Step 2:** When you click on the button “create task”, you’ll be taken to the next window (see figure 23), where you will be able to add a new task, by clicking on the button “new”.

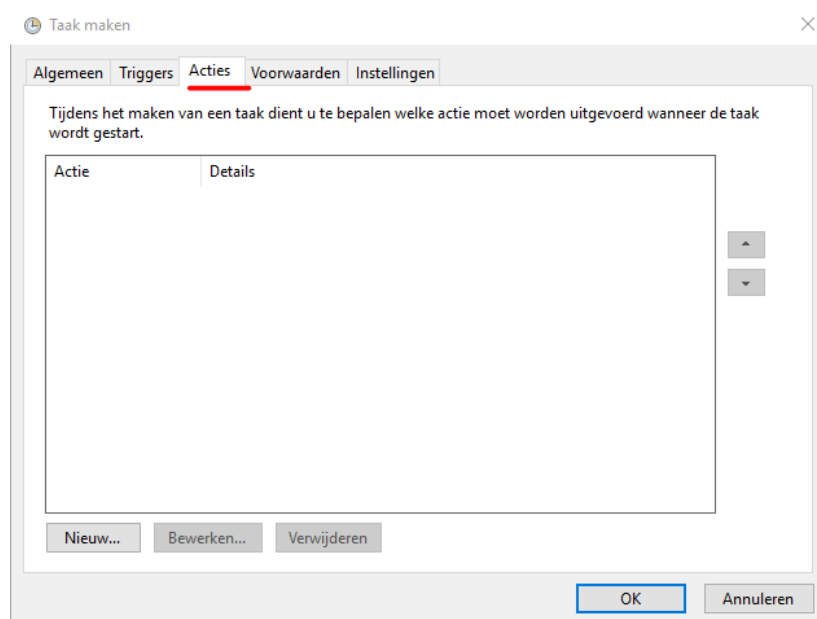


Figure 23

**Step 3:** After following step 2, you'll get to the next window (see figure 24), which will contain the actual setup of the task. There will be three different variables, which need to be assigned, in order for the program to execute the task properly.

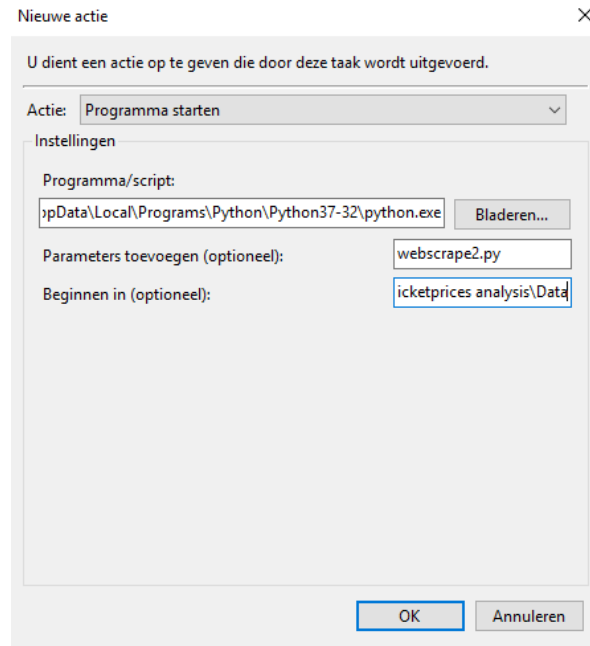


Figure 24

**Step 4:** First of all we need to define the location where Python3 is being located. We can do this by going to our cmd interface and type the following code: `python -c "import sys; print(sys.executable)"` (see figure 25). By executing this line of code, it will show the location where our Python3 program is being stored. Just copy and paste this code under "Program/script" (see figure 24) NOTE: is not the location of the script we have written in Python3, but the filelocation of the Software program Python3 itself.

```
Opdrachtprompt
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>python webscraper2.py
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
C:\Users\dirkj\Workspace\advanced-course-webprogramming\Dataset\Ticketprices analysis\Data>python -c "import sys; print(sys.executable)"
C:\Users\dirkj\AppData\Local\Programs\Python\Python37-32\python.exe
```

Figure 25

**Step 5:** After defining the location of Python3, we need to define the name of our script. In my case, I named my script webscraper2 so the full name will be webscraper2.py (".py" indicates that the file is a python file). Paste this file name under Add parameters as can be seen in figure 24.

**Step 6:** The last part of creating the task, is defining the location, where your script is being stored. You can find this location by simply going to the folder where you saved your script, and copy the location from there (see figure 26) When the location is copied, it can be pasted under “Start in”, as can be seen in figure 24.

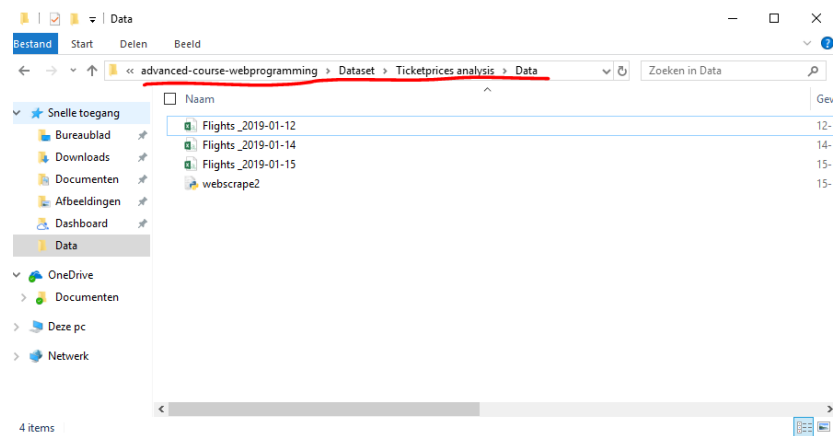


Figure 26

**Step 7:** After creating the task in step 1-6, we need to set up a trigger, which will execute the task, which we just created. To create a trigger, go to the “trigger” section and click on “new”. (see figure 27)

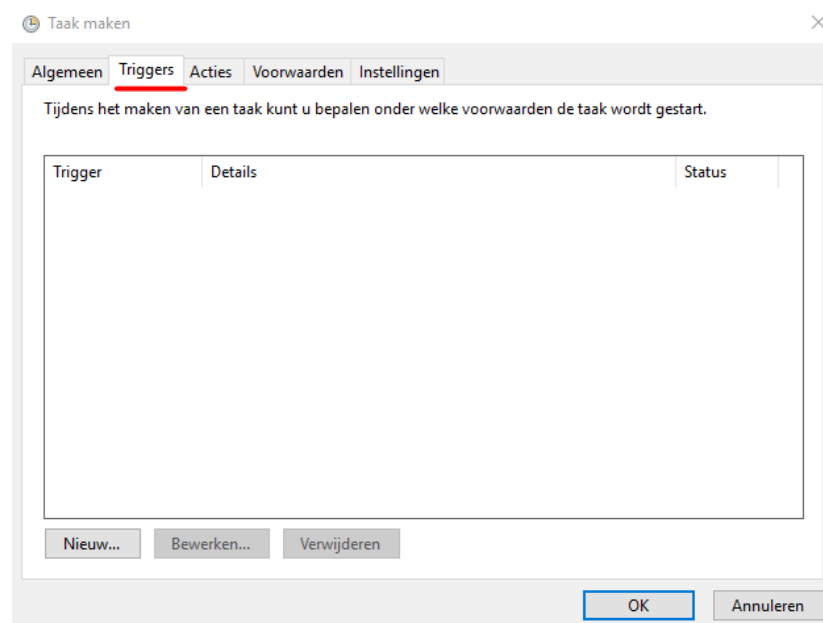


Figure 27

**Step 8:** When setting up the trigger, as can be seen in figure 28, we need to define a couple of things. First an foremost, we should define the frequency in which our script will be executed. In our case, the execution of the script will happen on a daily basis, which means we have to select “Daily”. After selecting “Daily” as our frequency of execution, we need to define a date on which the execution will start

and around what time the script will be executed. When this is all set, we can click on “ok” and our task is scheduled. (see figure 29)

Nieuwe trigger ✕

Start deze taak: Gepland

Instellingen

☐ Eenmalig ☒ Dagelijks ☐ Wekelijks ☐ Maandelijks

Start: 15- 1-2019 14:10:27 ☐ Sync. tussen tijdzones

Elke: 1 dag(en) uitvoeren

Geavanceerde instellingen

☐ Taak vertragen voor max. (willekeurig): 1 uur

☐ Taak herhalen elke: 1 uur gedurende: 1 dag

☐ Alle actieve taken aan einde van de herhalingsduur stoppen

☐ Taak stoppen indien actief langer dan: 3 dagen

☐ Verloopt op: 15- 1-2020 14:10:29 ☐ Sync. tussen tijdzones

☒ Ingeschakeld

OK Annuleren

Figure 28

Naam	Status	Triggers	Volgende keer uitvoeren	Vorige keer uitgevoerd	Resultaat van vorige keer uitvoeren	Auteur	Gemaakt
Avira_Antivir...	Gereed	Bij aanmelden van elke gebruiker		15-1-2019 13:31:19	De bewerking is voltooid. (0x0)	Avira	
GoogleUpda...	Gereed	Meerdere triggers opgegeven	16-1-2019 10:59:03	15-1-2019 10:59:03	De bewerking is voltooid. (0x0)		
GoogleUpda...	Gereed	Elke dag om 10:59 - Na trigger elke 1 uur herhalen gedurende 1 dag.	15-1-2019 14:59:03	15-1-2019 13:59:03	De bewerking is voltooid. (0x0)		
HPAudioSwi...	Actief	Bij aanmelden van elke gebruiker		12-1-2019 19:13:53	De taak wordt nu uitgevoerd. (0x41301)	HP Inc.	23-5-2016 17:1
HPCEASche...	Gereed	Om 14:19 op 19-1-2019 - Na trigger elke 06:00:00 herhalen gedurende 30.00:00:00.	19-1-2019 14:19:00	15-1-2019 14:19:00	De bewerking is voltooid. (0x0)	DESKTOP-SDS3169\dirkj	
HPEA3JOBS	Gereed	Bij aanmelden van elke gebruiker		14-6-2017 04:10:19	De bewerking is voltooid. (0x0)	WIN-R2D6H5F7ANA\Administrator	22-4-2017 04:5
HPJumpStar...	Actief	Bij aanmelden van elke gebruiker		12-1-2019 19:12:52	De taak wordt nu uitgevoerd. (0x41301)	HP Inc.	23-5-2016 17:1
Intel PTT EK ...	Gereed	Aangepast gebeurtenisfilter		12-1-2019 19:13:13	(0xFFFFFFFF)	Intel Corporation	8-9-2015 11:56
McAfeeLogon	Gereed	Bij aanmelden van elke gebruiker		12-1-2019 19:13:53	De bewerking is voltooid. (0x0)	McAfee	
Nvbackend_...	Gereed	Bij aanmelden van elke gebruiker		14-6-2017 04:34:39	De bewerking is voltooid. (0x0)		
NvNotifier_...	Gereed	Bij aanmelden van elke gebruiker		12-1-2019 19:14:53	De bewerking is voltooid. (0x0)	NVIDIA Corporation	
OneDrive St...	Gereed	Om 04:00 op 1-5-1992 - Na trigger elke 1.00:00:00 eindeloos herhalen.	16-1-2019 04:51:18	30-11-1999 00:00:00	De taak is nog niet uitgevoerd. (0x41303)	Microsoft Corporation	
OneDrive St...	Gereed	Om 04:00 op 1-5-1992 - Na trigger elke 1.00:00:00 eindeloos herhalen.	16-1-2019 06:53:23	14-1-2019 09:22:44	(0x8004EE04)	Microsoft Corporation	
Scheduler	Actief	Elke dag om 14:23	15-1-2019 14:23:27	30-11-1999 00:00:00	De taak is nog niet uitgevoerd. (0x41303)	DESKTOP-SDS3169\dirkj	15-1-2019 14:2

Figure 29