Tutorial for Data Preprocessing

The questions are as follows:

- How can we get data at once from a public website?
- How can we extract structured data from pdf files?

Download PDF Files

An inquiry example is as follows:

```
https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=1&product=&year=2019&challenge=0&method=getmapmarkers
```

When users input the year of interest, the website will return the inquired results. However, the website developer will require users to click marker one by one. Fortunately, we can try to get the data (in JSON format) at once with the following codes:

Python codes

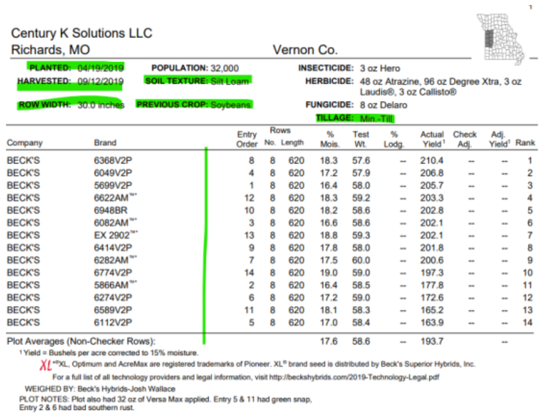
```
import os.path
import traceback
def do_something_wrong():
    cc = int("come on!")
    return cc

def capture_Error():

    try:  # first catch
        do_something_wrong()
    except Exception as ex:
        print(f"The Exception is here:\n{ex}")

    try:  # second catch
        do_something_wrong()
    except:
        print(f"Use traceback.format_exc() instead:\n{traceback.format_exc()}")
##
# The first catch would only display
# The Exception is here:
# invalid literal for int() with base 10: 'come on!'

#while the second catch includes not only the error but where it occurs
## the second catch includes not only the error but where it occurs
## In a large software project, the second example would be way more helpful than the first.

import requests
import csv

# The URL to send the request to
# Cron: 2019
#url =
"https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500
&family=1&product=&year=2019&challenge=0&method=getmapmarkers"

# Soybean: 2019
# url =
https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&f
amily=1&product=&year=2019&challenge=0&method=getmapmarkers

def parstItemsFromURL(url,yearStr):
   # Send a GET request to the URL
   response = requests.get(url)

   # Check if the request was successful
   if response.status_code == 200:
      # Parse the JSON response
      data = response.json()

      # Define the CSV file name
      csv_file = "TestPlotId_"+yearStr+".csv"

      # if os.path.exists(csv_file):
      #     os.remove(csv_file)
      # else:
      #     pass

      # Open the CSV file in write mode
      with open(csv_file, mode='a', newline='') as file:
         writer = csv.writer(file)

         # Write the header row
         writer.writerow(["TestPlotId", "Latitude", "Longitude", "Address", "Name",
"YearCode"])
```

```
            # Write the data rows
            for item in data:
                test_plot_id = item.get('TestPlotId')
                latitude = item.get('Latitude')
                longitude = item.get('Longitude')
                address = item.get('Address')
                name = item.get('Name')
                year_code = item.get('YearCode')

                # Write the data to the CSV file
                writer.writerow([test_plot_id, latitude, longitude, address, name, year_code])

        print(f"Data saved successfully to {csv_file}")
    else:
        print(f"Failed to retrieve data. Status code: {response.status_code}")

    return 1

# if __name__ == "__main__":
#     capture_Error()

def deletePreviousCSV(yearStr):
    csv_file = "test_plots_data_" + yearStr + ".csv"
    if os.path.exists(csv_file):
        os.remove(csv_file)
    else:
        pass

if __name__ == "__main__":
    print("Start downloading pdf files")
    # url =
"https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500
&family=1&product=&year=2019&challenge=0&method=getmapmarkers"

    urlBasicStr =
"https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500
&family=1&product=&year=2019&challenge=0&method=getmapmarkers"
    familyIDLis = ["1","2","3"]
    yearList = ["2019","2020","2021","2022"]
    # familyID = "1"
    # yearStr = "2020"
    for yearStr in yearList:
        deletePreviousCSV(yearStr)

    for yearStr in yearList:
        for familyID in familyIDLis:
```

```
        url =
"https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500
&family=" + familyID + "&product=&year=" + yearStr +
"&challenge=0&method=getmapmarkers"
        print("it is processing:{}".format(url))
        parstItemsFromURL(url, yearStr)
```

Digitalize PDF files

We can write python codes to extract useful information from pdf files according to the structure. For instance, we want to get the planting and harvesting dates from a pdf file.

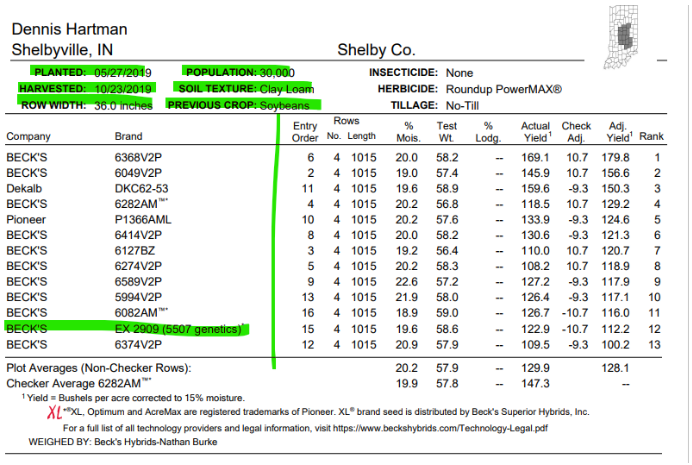


Century K Solutions LLC
Richards, MO   Vernon Co.

PLANTED: 04/19/2019	POPULATION: 32,000	INSECTICIDE:	3 oz Hero
HARVESTED: 09/12/2019	SOIL TEXTURE: Silt Loam	HERBICIDE:	48 oz Atrazine, 96 oz Degree Xtra, 3 oz Laudis®, 3 oz Callisto®
ROW WIDTH: 30.0 inches	PREVIOUS CROP: Soybeans	FUNGICIDE:	8 oz Delaro
		TILLAGE:	Min.-Till

Company	Brand	Entry Order	Rows No.	Rows Length	% Mois.	Test Wt.	% Lodg.	Actual Yield[1]	Check Adj.	Adj. Yield[1]	Rank
BECK'S	6368V2P	8	8	620	18.3	57.6	--	210.4	--	--	1
BECK'S	6049V2P	4	8	620	17.2	57.9	--	206.8	--	--	2
BECK'S	5699V2P	1	8	620	16.4	58.0	--	205.7	--	--	3
BECK'S	6622AM™*	12	8	620	18.3	59.2	--	203.3	--	--	4
BECK'S	6948BR	10	8	620	18.2	58.6	--	202.8	--	--	5
BECK'S	6082AM™*	3	8	620	16.6	58.6	--	202.1	--	--	6
BECK'S	EX 2902™*	13	8	620	18.8	59.3	--	202.1	--	--	7
BECK'S	6414V2P	9	8	620	17.8	58.0	--	201.8	--	--	8
BECK'S	6282AM™*	7	8	620	17.5	60.0	--	200.6	--	--	9
BECK'S	6774V2P	14	8	620	19.0	59.0	--	197.3	--	--	10
BECK'S	5866AM™*	2	8	620	16.4	58.5	--	177.8	--	--	11
BECK'S	6274V2P	6	8	620	17.2	59.0	--	172.6	--	--	12
BECK'S	6589V2P	11	8	620	18.1	58.3	--	165.2	--	--	13
BECK'S	6112V2P	5	8	620	17.0	58.4	--	163.9	--	--	14
Plot Averages (Non-Checker Rows):					17.6	58.6	--	193.7		--	

[1] Yield = Bushels per acre corrected to 15% moisture.
XL *®XL, Optimum and AcreMax are registered trademarks of Pioneer. XL® brand seed is distributed by Beck's Superior Hybrids, Inc.
For a full list of all technology providers and legal information, visit http://beckshybrids.com/2019-Technology-Legal.pdf
WEIGHED BY: Beck's Hybrids-Josh Wallace
PLOT NOTES: Plot also had 32 oz of Versa Max applied. Entry 5 & 11 had green snap, Entry 2 & 6 had bad southern rust.



Dennis Hartman
Shelbyville, IN   Shelby Co.

PLANTED: 05/27/2019	POPULATION: 30,000	INSECTICIDE:	None
HARVESTED: 10/23/2019	SOIL TEXTURE: Clay Loam	HERBICIDE:	Roundup PowerMAX®
ROW WIDTH: 36.0 inches	PREVIOUS CROP: Soybeans	TILLAGE:	No-Till

Company	Brand	Entry Order	Rows No.	Rows Length	% Mois.	Test Wt.	% Lodg.	Actual Yield[1]	Check Adj.	Adj. Yield[1]	Rank
BECK'S	6368V2P	6	4	1015	20.0	58.2	--	169.1	10.7	179.8	1
BECK'S	6049V2P	2	4	1015	19.0	57.4	--	145.9	10.7	156.6	2
Dekalb	DKC62-53	11	4	1015	19.6	58.9	--	159.6	-9.3	150.3	3
BECK'S	6282AM™*	4	4	1015	20.2	56.8	--	118.5	10.7	129.2	4
Pioneer	P1366AML	10	4	1015	20.2	57.6	--	133.9	-9.3	124.6	5
BECK'S	6414V2P	8	4	1015	20.0	58.2	--	130.6	-9.3	121.3	6
BECK'S	6127BZ	3	4	1015	19.2	56.4	--	110.0	10.7	120.7	7
BECK'S	6274V2P	5	4	1015	20.2	58.3	--	108.2	10.7	118.9	8
BECK'S	6589V2P	9	4	1015	22.6	57.2	--	127.2	-9.3	117.9	9
BECK'S	5994V2P	13	4	1015	21.9	58.0	--	126.4	-9.3	117.1	10
BECK'S	6082AM™*	16	4	1015	18.9	59.0	--	126.7	-10.7	116.0	11
BECK'S	EX 2909 (5507 genetics)*	15	4	1015	19.6	58.6	--	122.9	-10.7	112.2	12
BECK'S	6374V2P	12	4	1015	20.9	57.9	--	109.5	-9.3	100.2	13
Plot Averages (Non-Checker Rows):					20.2	57.9	--	129.9		128.1	
Checker Average 6282AM™*					19.9	57.8	--	147.3		--	

[1] Yield = Bushels per acre corrected to 15% moisture.
XL *®XL, Optimum and AcreMax are registered trademarks of Pioneer. XL® brand seed is distributed by Beck's Superior Hybrids, Inc.
For a full list of all technology providers and legal information, visit https://www.beckshybrids.com/Technology-Legal.pdf
WEIGHED BY: Beck's Hybrids-Nathan Burke

Python codes: to extract specific items

```
import pdfplumber
import re
  with pdfplumber.open(pdf_path) as pdf:
        # Assuming the information is on the first page
        page = pdf.pages[0]
        text = page.extract_text()
        print(text)

    # Regular expressions to find PLANTED and HARVESTED dates
    planted_match = re.search(r'PLANTED:\s*(\d{2}/\d{2}/\d{4})', text)
    harvested_match = re.search(r'HARVESTED:\s*(\d{2}/\d{2}/\d{4})', text)

    planted_date = planted_match.group(1).strip() if planted_match else None
    harvested_date = harvested_match.group(1).strip() if harvested_match else None

    # Use regular expression to find the content between start_keyword and end_keyword
    pattern = rf'{"SOIL TEXTURE"}:\s*(.*?)\s*{"HERBICIDE"}:'

    soil_texture_match = re.search(pattern, text, re.DOTALL)
    soil_texture = soil_texture_match.group(1).strip() if soil_texture_match else "Content not
found."
```

When we use python to digitize a pdf file, we will get the data organized irregularly. To solve this problem, for the table, we can read line by line and merge elements if necessary. More importantly, we need to carefully check the data structure and then write codes to organize it. An example is offered as follows:

Python codes: iterate through pdf files and extract information and save it into csv files.

```
###Creator: Xinghua Cheng
###Date: August 29, 2024
###
import glob
import pdfplumber
import re
import pandas as pd
import os

def merge_elements(lst):
    # Step 1: Find the index of '--'
    # try:
    #     o_index = lst.index('--')
    # except ValueError:
    #     raise ValueError("'--' not found in the list")
    #
    #
    o_index = 0
    try:
        # Find the index of the last '--'
        o_index = len(lst) - 1 - lst[::-1].index('--')
        print(f"The index of the last '--' is: {lst}")
    except ValueError:
        print("The list does not contain '--'.")

    # Step 2: Determine the range to merge (from the start to O-5)
    merge_start = 0
    merge_end = o_index - 6

    # Step 3: Merge the elements from merge_start to merge_end into a single string
    merged_element = '_'.join(lst[merge_start:merge_end + 1])

    # Step 4: Construct the new list
    new_list = [merged_element] + lst[merge_end + 1:]

    return new_list

def merge_companyBrand(lst):

    # Determine the range to merge (from the start to O-5)
    merge_start = 0
    merge_end = len(lst)-11

    # Step 3: Merge the elements from merge_start to merge_end into a single string
    merged_element = '_'.join(lst[merge_start:merge_end + 1])

    # Step 4: Construct the new list
    new_list = [merged_element] + lst[merge_end + 1:]

    return new_list

def getCropType(dfCropYield):
    cropType = "Corn"

    #get the column of Yield1 and get the average and take it as aveYield


    # Convert the 'yield1' column from strings to numbers
    Yield1 = dfCropYield['actual_yield1']
    Yield1
    dfCropYield['actual_yield1_numeric'] = pd.to_numeric(dfCropYield['actual_yield1'],
errors='coerce')

    # Calculate the average of the 'yield1' column
    average_yield = dfCropYield['actual_yield1_numeric'].mean()

    print("the average_yield is:{}".format(average_yield))
    #if aveYield >=100, it is Soybean

    #if aveYield >, it is Corn

    if average_yield>=100:
        cropType = "Corn"
    elif average_yield<100: # I just open pdf files and determine the value with empirical
experience
        cropType = "Soybean"
    else:
        cropType = "TBD_type"

    print("The Crop Type is:{}".format(cropType))

    print("To ensure the crop type is correct, please also go to check the planting date.")

    return cropType

#The function that aims to extract information from Beck PDF Files.
def extract_information_from_pdf(pdf_path):

    def count_items_in_line(line):
        # Split the line by spaces and count non-empty items
        items = [item for item in line.split() if item]
        return len(items)

    with pdfplumber.open(pdf_path) as pdf:
        # Assuming the information is on the first page
        page = pdf.pages[0]
        text = page.extract_text()
        print(text)
        # file1 = open("myfile.txt", "w")
        # file1.write(text)
        # file1.close()
        # Extract the table (assuming there's only one table on the page)

        # Extract the first page
        first_page = pdf.pages[0]

        # Split the extracted text into lines
        lines = text.splitlines()
        # Find the line where "Company" is included
        # company_line = next((line for line in lines if "Company" in line), "Company not
found.")
        # print(f"Line containing 'Company': {company_line}")
    # Find the index of the line where "Company" is included
    # We will got the case "Fertilzer Service Company" in the first line
    company_line_index = next((index for index, line in enumerate(lines) if "Brand" and
"Company" in line and count_items_in_line(line)>=5), -1)

    header_items_count = count_items_in_line(lines[company_line_index])

    value_lines = []
    if company_line_index != -1:
        # Get the number of items in the header line
        header_items_count = count_items_in_line(lines[company_line_index])

        # Extract all subsequent lines that correspond to the values for those fields

        for line in lines[company_line_index + 1:]:
            # Check if the line has the same number of items as the header line
            if count_items_in_line(line) >= header_items_count and "--" in line:
                value_lines.append(line)
            else:
                # Break if the line does not match the expected number of items
                break

        # Output the header and the corresponding value lines
        print(f"Header Line: {lines[company_line_index]}")
        print("Value Lines:")
        for value_line in value_lines:
            print(value_line)
    else:
        print("Line containing 'Company' not found.")

    # Split the header line into columns
    columns = [item for item in lines[company_line_index].split() if item]
    lenthColumns = len(columns)
    columnsNew = merge_companyBrand(columns)
    pathPDF, fileNamePDF = os.path.split(pdf_path)

    columnsNew.append('PDF_file')
    # Split each value line into a list of values
    data = []
    for value_line in value_lines:
        if True: #count_items_in_line(value_line) >= header_items_count:
            #we need to process it
            valueLis = [item for item in value_line.split() if item]
            lenthvalueLis = len(valueLis)
            # for instance:
            # BECK'S EX 2909 (5507 genetics)*
            # we need to change it to [BECK'S,EX_2909_(5507 genetics)*,16, 8 ,556, 20.2, 56.3, -
-, 214.5, -6.1, 208.4, 14]
            valueLisNew = []
            # if lenthvalueLis == lenthColumns:
            #     #we do not to need the first two column
            #     valueLisNew = valueLis
            #
            # else:
            valueLisNew = merge_companyBrand(valueLis)
            valueLisNew.append(pdf_path)
            data.append(valueLisNew)

    # we can take the maximum number of items as the number of cloumn, so we can save
result as csv
    # Create a DataFrame using the columns and data
    dfCropYield = pd.DataFrame(data, columns=columnsNew)


##################################################################################
###
    # Check with df and see if it is either Crop or Soybean

##################################################################################
###

    # Rename the columns
    columns = dfCropYield.columns.tolist()
    first_yield1_index = columns.index('Yield1')
    second_yield1_index = first_yield1_index + columns[first_yield1_index +
1:].index('Yield1') + 1

    dfCropYield.columns.values[first_yield1_index] = 'actual_yield1'
    dfCropYield.columns.values[second_yield1_index] = 'adj_yield1'

    # # Display the DataFrame
    print("data frame for the pdfFile:{}".format(pdf_path))
    print(dfCropYield)

    crop_type = getCropType(dfCropYield) #it is either Corn or Soybean in the U.S. Corn belt
    crop_type


##################################################################################
###
    # Save the information of crop yield and brand
```

```
####################################################################################
###

    crop_yield_brand_csv = pathPDF+"\\"+"Beck_plot_"+fileNamePDF[:-4]
+"_crop_yield_brand"+ "_"+ crop_type + ".csv"

    if os.path.exists(crop_yield_brand_csv):
        os.remove(crop_yield_brand_csv)
    else:
        pass

    dfCropYield.to_csv(crop_yield_brand_csv, index=False)

    print("the information of crop yield and brand is saved in
:{}".format(crop_yield_brand_csv))

####################################################################################
###
    # Extract information from the first "table" from a pdf file

####################################################################################
###

    # Regular expression to find the "County, State" pattern
    county_state_match = re.search(r'(\w+),\s+(\w{2})', text)
    county_state = county_state_match.group(0).strip() if county_state_match else None

    # Regular expressions to find PLANTED and HARVESTED dates
    planted_match = re.search(r'PLANTED:\s*(\d{2}/\d{2}/\d{4})', text)
    harvested_match = re.search(r'HARVESTED:\s*(\d{2}/\d{2}/\d{4})', text)

    planted_date = planted_match.group(1).strip() if planted_match else None
    harvested_date = harvested_match.group(1).strip() if harvested_match else None

    # Correct regular expression for PREVIOUS CROP
    previous_crop_match = re.search(r'PREVIOUS CROP:\s*(\w+)', text)
    previous_crop = previous_crop_match.group(1).strip() if previous_crop_match else None

    # Correct regular expression for tillage
    # tillage_match = re.search(r'TILLAGE:\s*(\w+)', text)
    # tillage = tillage_match.group(1).strip() if tillage_match else None

    # Use regular expression to find the content after "TILLAGE:"
    tillage_match = re.search(r'TILLAGE:\s*(.*)', text)
    tillage = tillage_match.group(1).strip() if tillage_match else "Content not found."
    insecticide_match = re.search(r'INSECTICIDE:\s*(.*)', text)
    insecticide = insecticide_match.group(1).strip() if insecticide_match else "Content not
found."

    # Use regular expression to find the content between start_keyword and end_keyword
    pattern = rf'{"SOIL TEXTURE"}:\s*(.*?)\s*{"HERBICIDE"}:'

    soil_texture_match = re.search(pattern, text, re.DOTALL)
    soil_texture = soil_texture_match.group(1).strip() if soil_texture_match else "Content not
found."


    pattern2 = rf'{"ROW WIDTH"}:\s*(.*?)\s*{"PREVIOUS CROP"}:'
    row_width_match = re.search(pattern2, text, re.DOTALL)
    row_width = row_width_match.group(1).strip() if row_width_match else "Content not
found."

    return county_state, planted_date,
harvested_date,previous_crop,tillage,soil_texture,insecticide,row_width,crop_type

###Goal: To determine whether this file was generated for Corn or Soybean.
def iterate_through_pdf_files(pdfFileInter):

    #Step 1: read pdf files and get the items of interest
    pdfFileInterLis = glob.glob(pdfFileInter+"\\"+"*.pdf")

    #pdfFileInterLis = [pdfFileInter+"\\"+"94338.pdf"]

    for pdfFile in pdfFileInterLis:
        path,fileName = os.path.split(pdfFile)

        print("The file: {}".format(pdfFile)+"is being processed:\n")
        # Example usage
        pdf_path = pdfFile #'pdf_downloads_2019/94456.pdf'

        County_State, PLANTED_DATE, HARVESTED_DATE, PREVIOUS_CROP,
TILLAGE, SOIL_TEXTURE, INSECTICIDE, ROW_WIDTH ,CROP_TYPE =
extract_information_from_pdf(pdf_path)

        print(f"County, State(abbreviation): {County_State}")
        print(f"PLANTED Date: {PLANTED_DATE}")
        print(f"HARVESTED Date: {HARVESTED_DATE}")
        print(f"PREVIOUS CROP: {PREVIOUS_CROP}")
        print(f"TILLAGE: {TILLAGE}")
        print(f"SOIL TEXTURE: {SOIL_TEXTURE}")
        print(f"INSECTICIDE: {INSECTICIDE}")
        print(f"ROW WIDTH: {ROW_WIDTH}")

        PDFCSV = pdfFile
        plotID = fileName[:-4] #94456.pdf -> 94456
        columnsCSV = ["County_State", "PLANTED_DATE", "HARVESTED_DATE",
"PREVIOUS_CROP", "TILLAGE", "SOIL_TEXTURE", "INSECTICIDE", "ROW_WIDTH"
,"CROP_TYPE","plotID"]
        data = [County_State, PLANTED_DATE, HARVESTED_DATE, PREVIOUS_CROP,
TILLAGE, SOIL_TEXTURE, INSECTICIDE, ROW_WIDTH ,CROP_TYPE,plotID]
        df = pd.DataFrame([data], columns = columnsCSV)

        savedResultsCSV = pdfFileInter + "\\" + "Beck_" + plotID + "_" +
"plan_harvest_managment_"+ CROP_TYPE + ".csv"

        if df.empty:
            print("")
        else:
            if os.path.exists(savedResultsCSV):
                os.remove(savedResultsCSV)
            else:
                pass
            #save results
            df.to_csv(savedResultsCSV, index=False)
            print("the information of management (planting, harvesting, and tillage etc)is saved in
:{}".format(savedResultsCSV))

    ###csv records
    ## pdf_plotid_#_pdf_crop_type.csv
    ## for example: Year_2019_Plotid_94141_Corn.csv
    ## County_State,PLANTED_DATE, HARVESTED_DATE, PREVIOUS_CROP,
TILLAGE, SOIL_TEXTURE, INSECTICIDE, ROW_WIDTH, PDFCSV, CROP_TYPE,

    #save results as a dataframe

    return 1

if __name__ == "__main__":

    mainPath = "C:\\Users\\xic23015\\OneDrive - University of
Connecticut\\pythonProject\\crop_phenology_500m_merge\\process_SG-FG-
images\\Postprocessing"
    yearLis = ["2019","2020","2021","2022"] #["2019","2020","2021","2022"]
    pdfFileFolder = "pdf_downloads_" #For example, "pdf_downloads_2019"
    for yearStr in yearLis:
        pdfFileFolder = "pdf_downloads_" + yearStr
        pdfFileInter = mainPath + "\\" + pdfFileFolder
        iterate_through_pdf_files(pdfFileInter)

    print("\n")
    print("Finished! no errors!")
    print("please go forward and synthesize all csv files!")
```