

Semesterarbeit Teil 2: Erstellung von Graphiken mit Matplotlib

Aufgabenstellung

Erstellen Sie eine Beispielsammlung von verschiedenen Arten von Graphiken mit Matplotlib; es sollte je ein Beispiel der folgenden Diagrammtypen berücksichtigt werden:

- Funktionsgraphen,
- Mehrere Funktionsgraphen in derselben Graphik
- Balkendiagramme,
- Tortendiagramme,
- Histogramme.
- Beschreiben Sie jeweils, wie die Diagramme erzeugt werden.

Stellen Sie dabei auch Daten dar, die Sie von CSV-Dateien eingelesen haben.

Python 2.7 und Python 3

Ich habe festgestellt, dass ich für die Semesterarbeiten 1a und 1b die Python Version 2.7 verwendet habe. Diese wurde in der PATH-Variable überschrieben (da ich diese bereits vorher installiert hatte) und somit habe ich nicht die Anaconda-Version verwendet, sondern die normale Python 2.7 Version. Für die Semesterarbeiten 1a und 1b war dies kein Problem, da ich nicht auf irgendwelche Bibliotheken zugreifen musste. Für die Semesterarbeit 2 verwenden wir die Bibliothek Matplotlib, welche mit Anaconda installiert wird. Entsprechend habe ich auch die Scripts aus den Semesterarbeiten 1a und 1b angepasst. Allgemeine Anpassungen:

- Klammern für die print-Funktion: «print '<TEXT>'» => «print('<TEXT>')»

Spezifische Anpassung für die «file.py»-Bibliothek, welche ich auch für diese Arbeit verwenden werde:

```
def convert_value_to_number_if_possible(value):
    if isinstance(value, list):
        return convert_array_to_number_array_if_possible(value)

    if value.isdigit():
        return float(value)

    return value
```

In Python 3 gibt die Funktion «float» einen «nan»-Wert zurück, falls es sich nicht um eine gültige Zahl handelt. deshalb überprüfe ich dies mit der Funktion «.isdigit()» auf dem Wert und wandle diesen nur um, falls dieser auch eine Zahl ist, sonst wird der Wert ohne Umwandlung zurückgegeben.

Und:

```
def convert_array_to_number_array_if_possible(input):
    return list(map(convert_value_to_number_if_possible, input))
```

In Python 3 gibt die «map»-Funktion ein «map»-Objekt zurück und nicht mehr eine Liste, deshalb wandle ich diese mit der «list»-Funktion noch in eine gültige Liste um.

Implementation

Struktur

Für diese Arbeit verwende ich folgende Scripts:

- «test.py»: Dieses Script dient zur Überprüfung der Implementationen.
- «file.py»: Dieses Script enthält die Funktionen zum Auslesen einer Datei und wurde gemäss Anmerkung (oben) aus der letzten Semesterarbeit übernommen.

File-Script

Auf die Funktionen aus der letzten Arbeit gehe ich nicht genauer ein. Zusätzlich habe ich jedoch noch die Funktion «convert_table_data_to_numpy_column_data_array» implementiert:

```
def convert_table_data_to_numpy_column_data_array(data_table, remove_head_row = True):
    if remove_head_row:
        data_table = data_table[1:]

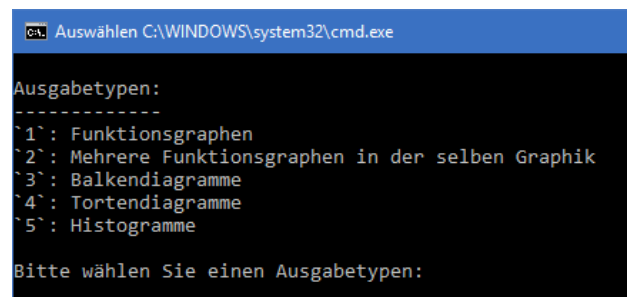
    return np.array(data_table).T
```

Diese Funktion wandelt die Liste, welcher aus der Datei ausgelesen wurde und eine 2-dimensionale Liste ist, welche nach Reihen und dann Spalten strukturiert ist, in eine 2-dimensionale Liste um, welche nach Spalten und dann Reihen strukturiert ist. Diese Funktion ist notwendig, da Matplotlib Datenwerte als Listen verwendet und nicht mit Tabellen, wie ich diese auslese, umgehen kann. Zusätzlich wandelt diese Funktion die Liste in eine numpy-Liste um, welche dann von Matplotlib verwendet wird. Zusätzlich kann man aus dem übergebenen Tabellen-Parameter die Heading-Reihe entfernen, da diese nicht als Datenwert verarbeitet werden kann, weil es sich hierbei um Text handelt.

Test-Script

Ich wusste nicht genau wie ich die verschiedenen Graphen aus der Aufgabenstellung genau darstellen sollte, deshalb habe ich mich dazu entschieden, dass man beim Scriptaufruf einen Ausgabetypen auswählen kann und dann der entsprechende Graph angezeigt wird. Wenn die Eingabe ungültig ist, wird das Script ohne Ausgabe geschlossen.

Die Daten für die Mehrheit der Graphen habe ich aus der Datei «fibonacci.csv» von der Semesterarbeit 1b ausgelesen. Diese Daten lese ich bei jedem Script-Aufruf aus.



```
C:\ Auswählen C:\WINDOWS\system32\cmd.exe

Ausgabetypen:
-----
^1: Funktionsgraphen
^2: Mehrere Funktionsgraphen in der selben Graphik
^3: Balkendiagramme
^4: Tortendiagramme
^5: Histogramme

Bitte wählen Sie einen Ausgabetypen:
```

Allgemein

Für die Darstellung des Graphen verwende ich folgende Bibliothek:

```
import matplotlib.pyplot as plt
```

Dann kann man bei den verschiedenen Ausgaben einfach das «plt»-Objekt konfigurieren und entsprechend anzeigen.

```
# Set the graph title
plt.title('<TITEL>')
```

um den Titel des Graphen zu setzen und

```
# Show the graph
plt.show()
```

um den Graphen anzuzeigen verwende ich bei jeder Ausgabe.

Funktionsgraphen

Für den Funktionsgraphen verwende ich die Daten aus der Tabelle und zeige eine gestrichelte Linie mit Punkte für die Werte und in der Farbe Grün an:

```
plt.plot(dataGraph[0], dataGraph[1], linestyle='dashed', marker='o', color='green')
```

Die Dokumentation zur verwendeten Funktion ist auf folgender Webseite zu finden:

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot

Mehrere Funktionsgraphen in derselben Graphik

Mehrere Funktionsgraphen in derselben Graphik können mit derselben Funktion ausgegeben werden:

```
plt.plot(dataGraph[0], dataGraph[1], 'ro--', dataGraph[0], dataGraph[2], 'b^--')
```

Hier werden einfach alle Daten in folgendem Format übergeben: «x1, f2, f1, ..., xn, yn, fn», wobei «f» für das Format, also die Darstellung der Linie, steht.

Die Dokumentation zur verwendeten Funktion ist auf folgender Webseite zu finden:

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot

Balkendiagramme

Für die Balkendiagramme verwende ich die Daten aus der Tabelle und zeige die Werte als grüne Balken an:

```
plt.bar(dataGraph[0], dataGraph[1], color='green')
```

Die Dokumentation zur verwendeten Funktion ist auf folgender Webseite zu finden:

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar

Tortendiagramme

Für die Tortendiagramme verwende ich eigene Daten, da hier die Datenstruktur anders ist und die Fibonacci-Daten keinen Sinn ergeben würden:

```
# Custom values and labels
values = [13, 7, 25, 2, 2]
labels = ['Bern', 'Freiburg', 'Zürich', 'Genf', 'Uri']

# Prepare the graph-data
plt.pie(values, labels=labels, startangle=90)
```

Die Dokumentation zur verwendeten Funktion ist auf folgender Webseite zu finden:

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.pie

Histogramme

Für die Histogramme verwende ich zufällig generierte Daten, da hier die Datenstruktur anders ist und die Fibonacci-Daten keinen Sinn ergeben würden:

```
x = np.random.normal(size=1000)
plt.hist(x, normed=True, bins=30, color='green')
```

Mit der numpy-Bibliothek generiere ich 1000 zufällige Werte, welche ich dann an die «hist»-Funktion übergebe.

Die Dokumentation zu den verwendeten Funktionen ist auf folgender Webseite zu finden:

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.normal.html>

http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist