

Modulabschlussprüfung Programmierung II

Repertorium

- div. Beispielaufgaben
- Modulabschlussprüfung

Kapitel 1 – Einführung git & GitHub

- Theorie

Kapitel 2 – Numerisches Python I

- Theorie
- Vorlesung
- Übung
- Beispielaufgaben (numpy / matplotlib)

Kapitel 3 – Objektorientierung, Teil 1

- Theorie
- Vorlesung
- Übung

Kapitel 4 – Objektorientierung, Teil 2

- Theorie
- Vorlesung
- Übung

Kapitel 5 – Objektorientierung, Teil 3 Theorie

- Vorlesung
- Übung
- Beispielaufgaben (class)

Kapitel 6 – GUI Programmierung, Teil 1

- Theorie
- Vorlesung
- Übung
- Zusammenfassung GUI

Kapitel 7 – GUI Programmierung, Teil 2

- Theorie
- Vorlesung
- Übung

Kapitel 8 – GUI Programmierung: QtDesigner

- Theorie
- Vorlesung
- Übung
- Beispielaufgaben (qt)

Kapitel 9 – Matplotlib & Qt

- Theorie
- Vorlesung
- Übung

Kapitel 10 – Projektionen und Vektordaten, Teil 1: Shapely

- Theorie
- Vorlesung
- Übung

Kapitel 11 – Folium & GeoPandas

- Übung
- Beispielaufgaben (panda)

Kapitel 12 – Projektionen & Vektordaten, Teil 2: cartopy

- Vorlesung
- Übung

Repertorium

Kapitel 1 – Einführung git & GitHub

Kapitel 2 - Numerisches Python I

Übung

Aufgabe 1

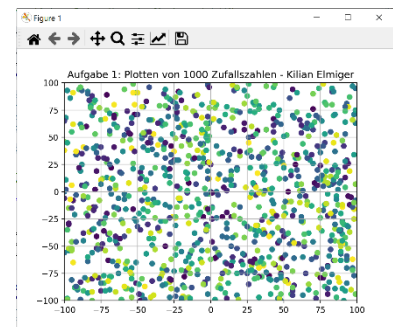
Plotten Sie 1000 Zufallszahlen im Bereich von $[-100, -100]$ bis $[100, 100]$ mit verschiedenen Farben.
Verwenden Sie dazu `matplotlib`.

```
n = 1000

x = np.random.randint(-100, 100, n)
y = np.random.randint(-100, 100, n)
rgb = np.random.randint(0, 256, n)

plt.scatter(x, y, c = rgb) #Ist eine Farbdefinition mit RGB überhaupt
möglich? - Ich erhalten bei der Schreibweise von (255,0,0) immer eine Fehlermeldung
plt.axis(xmin=-100, xmax=100, ymin=-100, ymax=100)
plt.grid(True)
plt.title("Aufgabe 1: Plotten von 1000 Zufallszahlen - Kilian Elmiger", fontsize=12)

plt.show()
```



Aufgabe 2:

Gegeben sei die Funktion $f(x, y)$:

$$f(x, y) = \exp(-x^2) \sin(y)$$

- Schreiben Sie die Funktion Numpy-kompatibel in Python
- Kreieren Sie nun einen Array mit den x-Koordinaten und y-Koordinaten, an denen f ausgewertet werden soll. Hierfür bietet sich `np.linspace(start, end, length)` an.
- Um aus den zwei eindimensionalen Arrays ein zweidimensionales Gitter zu machen, benutzen Sie `X, Y = np.meshgrid(x, y)`. Wenn x die Länge `len_x`, und y die Länge `len_y` hat, erzeugt `np.meshgrid` zwei 2D-Arrays der Grösse (len_y, len_x) . Jeder Eintrag der Arrays entspricht einem Punkt auf dem Gitter, welches durch x und y aufgespannt wird. X enthält alle x -Werte auf der jeweiligen GitterPosition, Y die entsprechenden y -Werte.
- Werten Sie nun $Z = f(X, Y)$. Mittels:

```
plt.pcolormesh(X, Y, Z)
plt.show()
```

können Sie $f(X, Y)$ zweidimensional grafisch darstellen.

Schreiben Sie Ihre Lösungen in die Datei `uebung2.py`. Unter Verwendung von passenden Dateinamen speichern Sie die Plots mit:

```
plt.savefig("passender_dateiname.png")
```

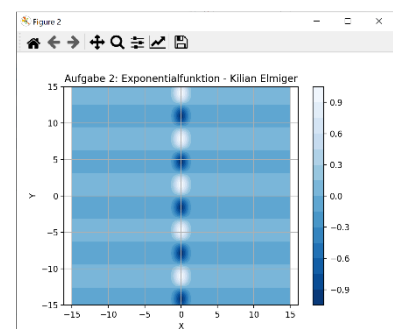
```
def f(x,y):
    return (np.exp(-x**2)*np.sin(y))

n = 256
x = np.linspace(-15, 15, n)
y = np.linspace(-15, 15, n)
X, Y = np.meshgrid(x, y)

figure_a2 = plt.figure(2)
supp_a2 = figure_a2.add_subplot(111)
plt.xlabel("X")
plt.ylabel("Y")
plt.axis("equal")
plt.grid(True)
plt.title("Aufgabe 2: Exponentialfunktion - Kilian Elmiger", fontsize=12)

img = supp_a2.contourf(X, Y, f(X,Y), 15, cmap='Blues_r')
figure_a2.colorbar(img)
plt.savefig("Exponentialfunktion_2d.png")

plt.show()
```



Kapitel 3 – Objektorientierung, Teil 1

[Übung](#)

Aufgabe 1

Erstellen Sie eine Klasse Vector3, welche einen dreidimensionalen Vektor repräsentiert.

Über den Konstruktor werden die Komponenten x, y und z definiert. Wird nichts angegeben, so wird ein Null-Vektor erstellt.

Entwickeln Sie eine Methode len, welche die Länge des Vektors berechnet.

Mit einer Instanz von Vector3 soll die Klasse getestet werden.

```
class Vector3:
    def __init__(self, x=0, y=0, z=0):
        self.x = x
        self.y = y
        self.z = z

    def len(self):
        return ((self.x**2)+(self.y**2)+(self.z**2))**0.5

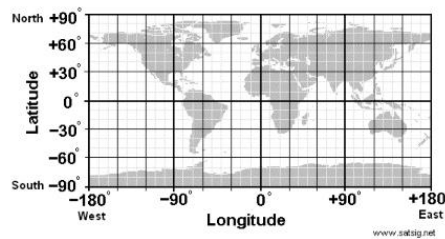
a = Vector3(1, 1, 1)
b = Vector3(2,2,2)
print(a.len()) ----- Output -----> 1.7320508075688772
print(b.len()) ----- Output -----> 3.4641016151377544
```

Aufgabe 2

Erstellen Sie eine Klasse WGS84Coord welche folgende Attribute hat:

`_longitude` (Länge)
`_latitude` (Breite)

latitude hat den Wertebereich $[-90, 90]$ und longitude hat den Bereich $[-180, 180]$.



Anforderungen:

- Im Konstruktor der Klasse kann die Länge und Breite übergeben werden. Der Standard-Wert ist 0.
- Stellen sie sicher, dass `_longitude` und `_latitude` immer im korrekten Wertebereich sind, also z.B. ist die Länge nie grösser als 180. Verwenden Sie dazu Setter- und Getter-Methoden. Falls ein Wert ausserhalb des zulässigen Bereichs gesetzt wird, so wird dieser korrigiert und eine Warnung wird ausgegeben.
- Verwenden Sie Property Attribute für die Länge und Breite.

```
class WGS84Coord:
    def __init__(self, longitude=0, latitude=0):
        self.setValueL(longitude)
        self.setValueB(latitude)

    def setValueL(self, l):
        if l > 180:
            l = -180 + (l - 180)
            print("Warnung: Der Längengrad ist ausserhalb des Definitionsbereichs, er wurde auf {l}° geändert.")

        elif l < -180:
            l = 180 - (l + 180)
            print("Warnung: Der Längengrad ist ausserhalb des Definitionsbereichs, er wurde auf {l}° geändert.")

        self._valueL = l

    def getValueL(self):
        return self._valueL

    def setValueB(self, b):
        if b > 90:
            b = -90 + (b - 90)
            print(f"Warnung: Der Längengrad ist ausserhalb des Definitionsbereichs, er wurde auf {b}° geändert.")

        elif b < -90:
            b = 90 - (b + 90)
            print(f"Warnung: Der Längengrad ist ausserhalb des Definitionsbereichs, er wurde auf {b}° geändert.")

        self._valueB = b

    def getValueB(self):
        return self._valueB

    longitude = property(getValueL, setValueL)
    latitude = property(getValueB, setValueB)

P1 = WGS84Coord(175, 20)
print(P1.longitude) ----- Output -----> 175
print(P1.latitude) ----- Output -----> 20
```

Kapitel 4 – Objektorientierung, Teil 2

Übung

Aufgabe 1:

Implementieren Sie eine Klasse `Vector3`, welche einen 3D Vektor repräsentiert. Der Konstruktor sollte einen Standardwert für x, y, z gleich 0 haben, sodass eine Instanzierung von `Vector3` ohne Angabe der x,y,z Werte einen *Null-Vektor* generiert. Dabei sollen Magische Methoden implementiert werden:

- Konversion zu Zeichenkette
- Vergleich zweier Vektoren (`vec1 == vec2`) soll True sein, wenn alle Komponenten gleich sind
- Addition
- Subtraktion
- komponentenweise Multiplikation (`Vector3 * Vector3`)
- Multiplikation mit Skalar: (`float * Vector3`) oder (`int * Vector3`) oder (`Vector3 * float`) oder (`Vector3 * int`)

```
a = Vector3(3,4,2)
b = Vector3(2,1,0)
c = a + b          # Addition
d = a - b          # Subtraktion
e = a * b          # Komponentenweise Multiplikation
f = a.dot(b)       # Skalarprodukt
g = a.cross(b)     # Kreuzprodukt
```

Implementieren sie zudem auch folgende Methoden:

- `cross(b)` Berechnung des Kreuzproduktes
- `dot(b)` Berechnung des Skalarproduktes
- `normalize()` Vektor normalisieren

```
class Vector3:
    def __init__(self, x = 0, y = 0, z = 0):
        self.x = x
        self.y = y
        self.z = z

    def __str__(self):
        return f"({self.x}, {self.y}, {self.z})"

    def __eq__(self, other):
        if self.x == other.x and \
            self.y == other.y and \
            self.z == other.z:
            return True
        else:
            return False

    def __add__(self, other):
        return Vector3(self.x + other.x, self.y + other.y, self.z + other.z)

    def __sub__(self, other):
        return Vector3(self.x - other.x, self.y - other.y, self.z - other.z)

    def __mul__(self, other):
        if type(other) == int or type(other) == float:
            return Vector3(self.x * other, self.y * other, self.z * other)
        else:
            return Vector3(self.x * other.x, self.y * other.y, self.z * other.z)

    def __rmul__(self, other):
        if type(other) == int or type(other) == float:
            return Vector3(self.x * other, self.y * other, self.z * other)
        else:
            return Vector3(self.x * other.x, self.y * other.y, self.z * other.z)

    def dot(self, other):
        return ((self.x * other.x) + (self.y * other.y) + (self.z * other.z))

    def cross(self, other):
        return Vector3((self.y * other.z - self.z * other.y), \
                       (self.z * other.x - self.x * other.z), \
                       (self.x * other.y - self.y * other.x))

    def normalize(self):
        abs_self = (self.x**2 + self.y**2 + self.z**2) ** 0.5
        try:
            return Vector3((self.x / abs_self), (self.y / abs_self), (self.z / abs_self))
        except ZeroDivisionError:
            print("Division durch Null nicht erlaubt, Berechnung nicht möglich.")
```

```
if __name__ == '__main__':
```

```
    a = Vector3(3.5,4,2)
    b = Vector3(2,1,0)
```

```
    print("a = b:", a == b)
    print("a = a:", a == a)
```

```
    c = a + b          # Addition
    print("c=", c)
```

```
    d = a - b          # Subtraktion
    print("d=",d)
```

```
    e = a * b          # Komponentenweise Multiplikation
    print("e=",e)
```

```
    f = a.dot(b)        # Skalarprodukt
    print("f=",f)
```

```
    g = a.cross(b)      # Kreuzprodukt
    print("g=",g)
```

```
    h = a.normalize()   # Vektor normalisieren
    print("h=", h)
```

Output:

```
a = b: False
a = a: True
c= (5.5, 5, 2)
d= (1.5, 3, 2)
e= (7.0, 4, 0)
f= 11.0
g= (-2, 4.0, -4.5)
h= (0.6163156344279367, 0.7043607250604991, 0.35218036253024954)
```


Kapitel 5 – Objektorientierung, Teil 3 Theorie

[Uebung](#)

Aufgabe 1:

Schreiben Sie die Klassen **Dreieck**, **Rechteck**, **Kreis** und **Polygon**. Diese Klassen werden von folgender Python Klasse vererbt:

```
class Figur:
    def __init__(self, name="Figur"):
        self.name = name

    def umfang(self):
        return 0

    def __str__(self):
        return self.name
```

- Das Attribut name ist eine Zeichenkette welche den Namen des jeweiligen Objektes enthält.
- Die Methode umfang soll für die jeweilige Figur korrekt implementiert werden
- Bei der Klasse Polygon soll mittels len() die Anzahl der Eckpunkte des Polygons zurückgegeben werden.
- Finden Sie geeignete Konstruktoren um die Figuren zu konstruieren
- __str__ soll die Figur mit allen Koordinaten sinnvoll beschreiben, z.B`
 - "Kreis M=(2.3,4.2) r=3.4"
 - "Rechteck (0,0) - (10,15)"

Hinweise:

- Die Figuren sind 2D
- Verwenden Sie die Klasse Punkt (punkt.py) um die Koordinaten der Figuren zu verwalten.
- Die Seiten bei Rechteck sind parallel zur jeweiligen Koordinaten-Achse.
- Wählen Sie geeignete **Konstruktoren**, z.B. bei Polygon sollen beliebig viele Ecken unterstützt werden.

punkt.py

```
class Punkt:
    def __init__(self, x=0, y=0):
        if isinstance(x, (int, float)) and isinstance(y, (int, float)):
            self.x = x
            self.y = y
        else:
            raise ValueError("Die x,y-Koordinaten müssen reale Zahlen sein!")

    def __str__(self):
        return f'Punkt({self.x},{self.y})'

    def entfernung(self, other):
        if isinstance(other, Punkt):
            return ((self.x - other.x)**2 + (self.y - other.y)**2)**0.5
        raise NotImplementedError

    def __eq__(self, other):
        if isinstance(other, Punkt):
            return self.x == other.x and self.y == other.y
        elif isinstance(other, (list, tuple)) and len(other) == 2:
            return self.x == other[0] and self.y == other[1]
        else:
            raise NotImplementedError("Kann einen Punkt-Objekt nur mit einem anderen, oder mit einer (x,y) Liste / Tuple vergleichen")
```

```
import math
from typing import Type
from punkt import Punkt

class Figur:
    def __init__(self, name='Figur'):
        self.name = name

    def umfang(self):
        return 0
        raise NotImplementedError

    def flaeche(self):
        return 0
        raise NotImplementedError

    def __str__(self):
        return self.name

# YOUR CODE HERE ### -----

class Dreieck(Figur):
    def __init__(self, A, B, C):
        super().__init__("Dreieck")

        if type(A) != Punkt or \
            type(B) != Punkt or \
            type(C) != Punkt:
            raise TypeError("A, B und C müssen Klasse Punkt sein.")

        self.Ecke1 = A
        self.Ecke2 = B
        self.Ecke3 = C

    def umfang(self):
        s1 = ((self.Ecke2.x - self.Ecke1.x)**2 +
              (self.Ecke2.y - self.Ecke1.y)**2) ** 0.5
        s2 = ((self.Ecke3.x - self.Ecke2.x)**2 +
              (self.Ecke3.y - self.Ecke2.y)**2) ** 0.5
        s3 = ((self.Ecke1.x - self.Ecke3.x)**2 +
              (self.Ecke1.y - self.Ecke3.y)**2) ** 0.5

        return round((s1 + s2 + s3), 4)

    def flaeche(self):
        a = ((self.Ecke2.x - self.Ecke1.x)**2 +
              (self.Ecke2.y - self.Ecke1.y)**2) ** 0.5
        b = ((self.Ecke3.x - self.Ecke2.x)**2 +
              (self.Ecke3.y - self.Ecke2.y)**2) ** 0.5
        c = ((self.Ecke1.x - self.Ecke3.x)**2 +
              (self.Ecke1.y - self.Ecke3.y)**2) ** 0.5

        s = (a + b + c) / 2
        f = (s * (s-a) * (s-b) * (s-c)) ** 0.5

        return round(f, 4)

    def __str__(self):
        return f"\n{self.name}: \tEcken: {self.Ecke1}, {self.Ecke2}, {self.Ecke3}\n\
        \tFläche: {self.flaeche()} m{chr(178)}\n\
        \tUmfang: {self.umfang()} m"

class Kreis(Figur):
    def __init__(self, M, r):
        super().__init__("Kreis")

        if type(M) != Punkt:
            raise TypeError("M muss Klasse Punkt sein.")

        self.Mittelpunkt = M
        self.Radius = r

    def umfang(self):
        return round((2 * self.Radius * math.pi), 4)

    def flaeche(self):
        return round((self.Radius**2 * math.pi), 4)

    def __str__(self):
        return f"\n{self.name}: \t\tMittelpunkt: {self.Mittelpunkt}, Radius: {self.Radius} m\n\
        \tFläche: {self.flaeche()} m{chr(178)}\n\
        \tUmfang: {self.umfang()} m"
```

```

class Rechteck(Figur):
    def __init__(self, A, C):
        super().__init__("Rechteck")

        if type(A) != Punkt or \
            type(C) != Punkt:
            raise TypeError("A und C müssen Klasse Punkt sein.")

        self.Ecke1 = A
        self.Ecke3 = C

    def umfang(self):
        l1 = (self.Ecke3.x - self.Ecke1.x)
        l2 = (self.Ecke3.y - self.Ecke1.y)

        return round((abs(l1) + abs(l2)) * 2, 4)

    def flaeche(self):
        l1 = (self.Ecke3.x - self.Ecke1.x)
        l2 = (self.Ecke3.y - self.Ecke1.y)
        return round((abs(l1) * abs(l2)), 4)

    def __str__(self):
        return f"\n{self.name}: \tEcken: A-{self.Ecke1}, C-{self.Ecke3}\n\
\tFläche: {self.flaeche()} m{chr(178)}\n\
\tUmfang: {self.umfang()} m"

class Polygon(Figur):
    def __init__(self, *args):
        super().__init__("Polygon")

        self.args = args

        for i in range(len(self.args)-1):
            if type(self.args[i]) != Punkt:
                raise ValueError(
                    "Eingegebene Variablen müssen Klasse Punkt sein")

    def umfang(self):
        if len(self.args) <= 2:
            raise ValueError(
                "Zu wenige parameter wurden definiert, berechnung kann nicht durchgeführt wreden")

        else:
            u = 0
            laenge = len(self.args)
            u += ((self.args[0].x - self.args[laenge-1].x)**2 +
                (self.args[0].y - self.args[laenge-1].y)**2) ** 0.5
            for i in range(len(self.args)-1):
                u += ((self.args[i].x-self.args[i+1].x) ** 2 +
                    (self.args[i].y-self.args[i+1].y) ** 2) ** 0.5
            return round(u, 4)

    def flaeche(self):
        f = 0
        laenge = int(len(self.args))
        f += self.args[0].x * (self.args[laenge-1].y - self.args[1].y)
        f += self.args[laenge-1].x * (self.args[laenge-2].y - self.args[0].y)
        for i in range(1, len(self.args)-1):
            f += self.args[i].x * (self.args[i+1].y - self.args[i-1].y)
        return round((f/2), 4)

    def __str__(self):
        laenge = len(self.args)
        plist = []
        for i in range(0, laenge):
            plist += [f"Polygonpunkt {i+1}: {self.args[i]}"]

        return f"\n{self.name}: \t{plist}\n\
\tAnzahl Punkte: {laenge}\n\
\tFläche: {self.flaeche()} m{chr(178)}\n\
\tUmfang: {self.umfang()} m"

# -----
if __name__ == '__main__':

    d = Dreieck(Punkt(0, 0), Punkt(1, 1), Punkt(2, 0))
    print(d)

    k = Kreis(Punkt(0, 0), 1)
    print(k)

    r = Rechteck(Punkt(0, 0), Punkt(2, 2))
    print(r)

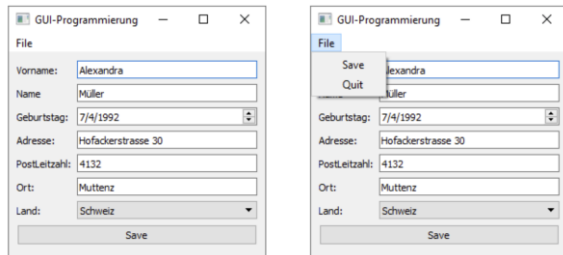
    p = Polygon(Punkt(0, 0), Punkt(1, 0), Punkt(1, 1), Punkt(2, 0), Punkt(
        2, 1), Punkt(2, 2), Punkt(3, 0), Punkt(3, 1), Punkt(3, 2), Punkt(3, 3))
    print(p)
    
```

Kapitel 6 – GUI Programmierung, Teil 1

Übung

Aufgabe 1

Erstellen Sie eine Applikation mit folgendem GUI:



- Implementieren Sie das GUI wie abgebildet, wählen Sie ein geeignetes Layout
- Ein File-Menu mit den Einträgen Save und Quit soll hinzugefügt werden
- wird auf den Button "Save" gedrückt, so wird ein File output.txt angelegt, welches die Daten kommagetrennt speichert, also für oberes Beispiel wäre der Inhalt:
``Alexandra,Müller,07/04/1992,Hofackerstrasse 30,4132,Muttenz,Schweiz``
- Beim Betätigen des "Quit" Menus wird das Programm beendet
- Beim Betätigen des "Save" Menus wird der Datensatz wie in c) gespeichert.

Hinweis

Die QComboBox (Auswahl Land) kann folgendermassen erstellt werden:

```
countries = QComboBox()
countries.addItem(["Schweiz", "Deutschland", "Österreich"])
```

Ausgelesen werden kann die QComboxBox mittels ``currentText()``:

```
land = countries.currentText()
```

```

from PyQt5 import QtGui
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from time import time

class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("GUI-Programmierung")

        # Layout -----
        layout = QFormLayout()

        # menubar -----
        menubar = self.menuBar()
        filemenu = menubar.addMenu("File")
        viewmenu = menubar.addMenu("View")

        save = QAction("Save", self)
        quit = QAction("Quit", self)

        filemenu.addAction(save)
        filemenu.addSeparator()
        filemenu.addAction(quit)

        quit.setMenuRole(QAction.QuitRole)

        save.triggered.connect(self.doSave)
        quit.triggered.connect(self.doQuit)

        # Widgets erstellen -----

        # Q LineEdit
        self.vorname = QLineEdit()
        self.name = QLineEdit()
        self.adresse = QLineEdit()
        self.plz = QLineEdit()
        self.ort = QLineEdit()

        # Q DateEdit
        self.geburtstag = QDateEdit(QDate.currentDate())

        # Q Combobox
        self.land = QComboBox()
        self.land.addItem(
            ["Schweiz", "Deutschland", "Österreich", "Frankreich", "Italien"])

        # Button
        save_button = QPushButton("Save")

        # Widgets dem Layout hinzufügen -----
        layout.addRow('Vorname:', self.vorname)
        layout.addRow('Name:', self.name)
        layout.addRow('Geburtstag', self.geburtstag)
        layout.addRow('Adresse:', self.adresse)
        layout.addRow('Plz:', self.plz)
        layout.addRow('Ort:', self.ort)
        layout.addRow('Land:', self.land)
        layout.addRow(save_button)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

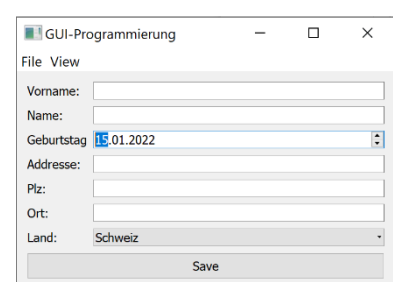
        # Eintrag speichern -----
        def doSave(self):
            sv_vorname = self.vorname.text()
            sv_name = self.name.text()
            sv_geburtstag = self.geburtstag.text()
            sv_adresse = self.adresse.text()
            sv_plz = self.plz.text()
            sv_ort = self.ort.text()
            sv_country = self.land.currentText()

            return f"{sv_vorname},{sv_name},{sv_geburtstag},{sv_adresse},{sv_plz},{sv_ort},{sv_country}\n"

        def doQuit(self):
            exit()

        # -----
        if __name__ == '__main__':

            app = QApplication([])
            fenster = Window()
            fenster.raise_()
            app.exec_()
    
```



Kapitel 7 – GUI Programmierung, Teil 2

Übung

Aufgabe 1

Erweitern Sie das GUI der letzten Aufgaben um folgendes:

Speichern

Verändern Sie das Verhalten bei "Speichern":

Es soll ein **File Dialog** verwendet werden um die Datei zu speichern.

Auf Karte zeigen

Mit dem Button "Auf Karte zeigen" oder über das Menu "View/Karte..." wird ein Webbrowser mit Google Maps geöffnet. Der Link kann mithilfe der Adresse zusammengesetzt werden, z.B:

<https://www.google.ch/maps/place/Hofackerstrasse+30+4132+MuttENZ+Schweiz>

in PyQt5 wird der Standard-Webbrowser folgendermassen geöffnet:

```
link = "http://www.fhnw.ch"
QDesktopServices.openUrl(QUrl(link)) # benötigt QtCore & QtGui
```

Vorsicht auch bei Sonderzeichen in der URL. Query Parameter (in unserem Fall die Adresse mit den +) können mit der urllib codiert werden:

```
import urllib.parse
query = 'Hello World@'
a = urllib.parse.quote(query). # enthält 'Hello%C3%B6%20W%C3%B6rld%40'
```

Laden

Fügen Sie den Button "Laden" hinzu. Dieser lädt einen zuvor gespeicherten Datensatz und stellt die Daten im GUI dar.

- Verwenden Sie dazu einen File-Dialog.
- Fügen Sie auch einen Menu-Eintrag zum Laden des Datensatzes hinzu.

Hinweis: Das QDateEdit (Beispiel: self.dateEdit) kann folgendermassen aus Text ausgefüllt werden:

```
# die Variable text enthält das Datum als Zeichenkette

dformat = QLocale().dateFormat(format=QLocale.FormatType.ShortFormat)
self.dateEdit.setDate(QDate.fromString(text, dformat))
```

```

from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *

class Window (QMainWindow):
    def __init__ (self):
        super().__init__()

        self.setWindowTitle("GUI-Programmierung")

#----- LAYOUT -----
        layout = QFormLayout()

        self.vorname = QLineEdit()
        self.nachname = QLineEdit()
        self.geburtstag = QDateEdit()
        self.adresse = QLineEdit()
        self.plz = QLineEdit()
        self.ort = QLineEdit()
        self.land = QComboBox()
        maps_button = QPushButton("Auf Karte anzeigen")
        load_button = QPushButton("Laden")
        save_button = QPushButton("Speichern")

        layout.addRow("Vorname:", self.vorname)
        layout.addRow("Nachname:", self.nachname)
        layout.addRow("Geburtstag:", self.geburtstag)
        layout.addRow("Adresse:", self.adresse)
        layout.addRow("Postleitzahl:", self.plz)
        layout.addRow("Ort:", self.ort)
        layout.addRow("Land:", self.land)
        layout.addRow(maps_button)
        layout.addRow(load_button)
        layout.addRow(save_button)

        self.land.addItem(["Schweiz", "Deutschland", "Österreich"])
        land = self.land.currentText()

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

#----- MENU -----
        menubar = self.menuBar()

        filemenu = menubar.addMenu("File")
        viewmenu = menubar.addMenu("View")

        save = QAction("Save", self)
        quit = QAction("Quit", self)
        load = QAction("Load", self)
        maps = QAction("Karte", self)

        quit.setMenuRole(QAction.QuitRole)

        filemenu.addAction(save)
        filemenu.addAction(load)
        filemenu.addSeparator() # Trennstrich
        filemenu.addAction(quit)
        viewmenu.addAction(maps)

        save.triggered.connect(self.doSave)
        load.triggered.connect(self.doLoad)
        quit.triggered.connect(self.doQuit)
        maps.triggered.connect(self.doMap)

#----- BUTTONS MIT FUNKTION VERBINDEN -----
        maps_button.clicked.connect(self.doMap)
        load_button.clicked.connect(self.doLoad)
        save_button.clicked.connect(self.doSave)

        self.show()
    
```

```
#----- FUNKTIONEN -----
```

```
def doMap(self):
    link = "https://www.google.ch/maps/place/"
    adresseingabe = self.adresse.text() + "+" + self.plz.text() + "+" + self.ort.text() + "+" +
self.land.currentText()
    QDesktopServices.openUrl(QUrl(link + adresseingabe))

def doLoad(self):
    filename, filter = QFileDialog.getOpenFileName(self, "Datei öffnen", "", "Textfile (*.txt)")
    if filename != "":
        file = open(f"{filename}", "r", encoding="utf-8")
        for zeile in file:
            element = zeile.split(",")

            self.vorname.insert(element[0])
            self.nachname.insert(element[1])
            datum = element[2]
            dformat = QLocale().dateFormat(format=QLocale.FormatType.ShortFormat)
            self.geburtstag.setDate(QDate.fromString(datum, dformat))
            self.adresse.insert(element[3])
            self.plz.insert(element[4])
            self.ort.insert(element[5])
            self.land.setCurrentText(element[6])

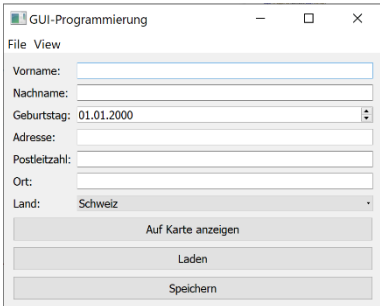
        file.close()

def doSave(self):
    filename, filter = QFileDialog.getSaveFileName(self, "Datei speichern", "", "Textfile (*.txt)")
    file = open(filename, "w", encoding="utf-8")
    file.write(self.vorname.text() + ',' + self.nachname.text() + ',' + self.geburtstag.text() + ',' +
self.adresse.text() + ',' + self.plz.text() + ',' + self.ort.text() + ',' + self.land.currentText())
    file.close()

def doQuit(self):
    exit(0)

# -----

app = QApplication([])
fenster = Window()
app.exec()
```



GUI-Programmierung

File View

Vorname:

Nachname:

Geburtstag: 01.01.2000

Adresse:

Postleitzahl:

Ort:

Land: Schweiz

Kapitel 8 – GUI Programmierung: QtDesigner

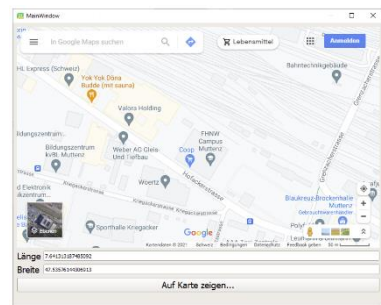
Uebung

Aufgabe 1

- Erstellen Sie mit dem Qt-Designer das unten abgebildete GUI mit Namen showmap.ui. Dabei soll für alle Widgets die Font-Grösse 14 verwendet werden
- Mit Klick auf den Button "Auf Karte zeigen..." soll Google Maps angezeigt werden. Der Link kann mithilfe der Koordinate zusammengesetzt werden. Das Format dazu ist: <https://www.google.ch/maps/place/breite,länge>
- Ein konkretes Beispiel wäre: <https://www.google.ch/maps/place/47.534874,7.642013>
- (optional) fügen Sie weitere Buttons hinzu welche für gewisse Orte automatisch die Länge Breite ausfüllen und auf dem Webbrowser angezeigt werden.

Hinweis: um die Datenschutzbestimmungen von Google nicht jedes mal anzuzeigen können Cookies persistent gespeichert werden:

```
defaultProfile = QWebEngineProfile.defaultProfile()
defaultProfile.setPersistentCookiesPolicy(QWebEngineProfile.ForcePersistentCookies)
```



```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import *
from PyQt5.uic import *
import numpy

# -----
class Browser(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("uebung-8-nadja pfister/showmap.ui", self)

        self.setWindowTitle("Uebung8 - Nadja Pfister")

        defaultProfile = QWebEngineProfile.defaultProfile()
        defaultProfile.setPersistentCookiesPolicy(QWebEngineProfile.ForcePersistentCookies)

        self.createConnects()
        self.show()

# -----
    def createConnects(self):
        self.KarteButton.clicked.connect(self.loadPage)
        self.randomButton.clicked.connect(self.random)

    def loadPage(self):
        Laenge = self.LaengelineEdit.text()
        Breite = self.BreitelineEdit.text()

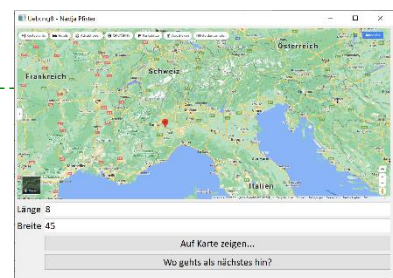
        if -90 <= float(Laenge) <= 90 and -180 <= float(Breite) <= 180:
            self.webEngineView.load(QUrl(f"https://www.google.ch/maps/place/{Breite},{Laenge}"))
        else:
            self.LaengelineEdit.setText("ungültiger Eingabe, Wert muss zwischen -180 und +180 liegen")
            self.BreitelineEdit.setText("ungültiger Eingabe, Wert muss zwischen -90 und +90 liegen")

    def random(self):
        randomL = "{0:.10}".format(numpy.random.uniform(-90, 90))
        randomB = "{0:.10}".format(numpy.random.uniform(-180, 180))

        self.LaengelineEdit.setText(randomL)
        self.BreitelineEdit.setText(randomB)

        self.webEngineView.load(QUrl(f"https://www.google.ch/maps/place/{randomB},{randomL}"))

# -----
app = QApplication([])
fenster = Browser()
app.exec()
```



Kapitel 9 – Matplotlib & Qt

Uebung

Aufgabe 1

Mit der numpy Funktion `np.poly1d` kann ein Polynom erstellt werden. Als Parameter werden die Koeffizienten in einer Liste definiert, dessen Länge definiert also den Grad.

Beispiele:

```
np.poly1d([7,3]) # f(x)=7x+3
np.poly1d([2,3,4]) # f(x)=2x^2+3x+4
```

Weiteres in der numpy Dokumentation:

<https://numpy.org/doc/stable/reference/generated/numpy.poly1d.html>

Das Resultat kann direkt geplottet werden, Beispiel:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
f = np.poly1d([2,3,4])
x = np.linspace(0,10,30)
y = f(x)
plt.plot(x,y,'ko-')
plt.show()
```

Erstellen Sie mit PyQt5 ein geeignetes GUI um allgemeine Polynome n-ten Grades einzugeben und darzustellen.

Tipp: Für die Koeffizienten könnte z.B. ein QLineEdit verwendet werden und die Daten kommagetrennt eingetippt.

Überprüfen Sie Eingaben auf Fehler.

Im GUI soll der Wertebereich und die Anzahl dargestellter Punkte definiert werden können. Verwenden Sie dazu geeignete Widgets.

Die Farbe des Plots soll ausgewählt werden können. Verwenden Sie dazu ein geeignetes Widget, z.B. eine QCombo Box

(Qt-Designer kann verwendet werden, muss aber nicht)

```
from PyQt5.QtWidgets import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
import matplotlib.pyplot as plt
import numpy as np
```

```
class Window(QMainWindow):
```

```
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Uebung 09 Plot in PyQt")

        layout = QVBoxLayout()
        figure = plt.figure(figsize=(16,9))

        # Widgets erstellen
        self.canvas = FigureCanvas(figure)
        self.f = QLineEdit("1,0,0")
        self.x = QLineEdit("-5, 5, 20")
        self.color = "r"

        self.labelf = QLabel()
        self.labelx = QLabel()
        self.labelsset()

        self.button = QPushButton("Plot")
        self.cb = QComboBox()
        self.cb.addItem(["b", "g", "r", "c", "y", "m", "k"])

        # Aktionen
        self.button.clicked.connect(self.plot)
        self.cb.currentIndexChanged.connect(self.selectionchange)

        # Widgets add to Layout
        layout.addWidget(self.canvas)
        layout.addWidget(self.labelf)
        layout.addWidget(self.f)
        layout.addWidget(self.labelx)
        layout.addWidget(self.x)
        layout.addWidget(self.button)
        layout.addWidget(self.cb)
```

```

# Hauptlayout
center = QWidget()
center.setLayout(layout)
self.setCentralWidget(center)
self.show()

# Farbwechsel
def selectionchange(self):
    self.color = self.cb.currentText()
    self.plot()

def labelsset(self):
    self.labelf.setText("Definiere dein Polygon")
    self.labelX.setText("Range: [ - , - ] Anz. Pkt.: -")

def labelupdate(self):
    inputf = self.f.text().split(",")

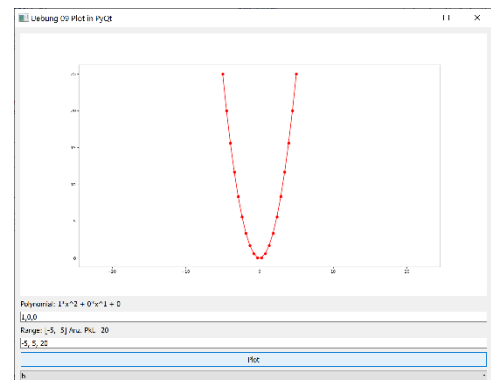
    polynomial_text = "Polynomial: "
    for i in range(len(inputf)-1):
        polynomial_text += f"{inputf[i]}*x^{(len(inputf)-1)-i} + "
    polynomial_text += inputf[-1]
    self.labelf.setText(polynomial_text)

    inputx = self.x.text().split(",")
    self.labelX.setText("Range: [" + inputx[0] + ", " + inputx[1] + "] Anz. Pkt. " + inputx[2])

def plot(self):
    plt.clf()
    try:
        f = eval(f"np.poly1d([{self.f.text()}])")
        x = eval(f"np.linspace({self.x.text()})")
        y = f(x)
    except:
        QMessageBox.warning(self, "Error", "Bitte gib die Daten richtig ein")
        self.f.setText("2,3,4")
        self.x.setText("0, 10, 30")
        self.labelsset()
    else:
        plt.plot(x, y, self.color+"o-")
        plt.axis("equal")
        self.canvas.draw()
        self.labelupdate()

app = QApplication([])
fenster = Window()
app.exec()

```



Kapitel 10 – Projektionen und Vektordaten, Teil 1: Shapely

Uebung

Aufgabe 1:

Erweitern Sie die Klasse `WGS84Coord` aus der Uebung 3:
 Zwei neue Methoden sollen hinzugefügt werden:
`toLV95()`: gibt eine die Koordinate als LV95 zurück (E,N)
`fromLV95(E,N)`: setzt die WGS84 Koordinate aus LV95
 Es soll pyproj verwendet werden.

```
from pyproj import Transformer

def check_longitude(val):
    if (val > 180) or (val < -180):
        print(f"The longitude value {val} either too small or too large."
              " Wrapping it into the interval [-180, 180]")

    # modulo trick
    new_val = (180 + val) % 360 - 180

    # by the above formula the values for 180, 3*180, 5*180, ...
    # are always -180 instead of 180 (which in this case is the same anyway)
    # we can fix these special cases:
    div, mod = divmod(val, 180)
    if (val > 0) and (mod == 0) and (div % 2 == 1): # check: div is odd and no remainder (mod = 0)
        new_val = 180

    return new_val

def check_latitude(val):
    if (val > 90) or (val < -90):
        print(f"The latitude value {val} either too small or too large."
              " Wrapping it into the interval [-90, 90]")

    new_val = (90 + val) % 180 - 90

    div, mod = divmod(val, 90)
    if (val > 0) and (mod == 0) and (div % 2 == 1):
        new_val = 90

    return new_val

class WGS84Coord:
    def __init__(self, longitude=0, latitude=0):
        self.longitude = longitude
        self.latitude = latitude

    def _set_longitude(self, longitude):
        self._longitude = check_longitude(longitude)

    def _get_longitude(self):
        return self._longitude

    def _set_latitude(self, latitude):
        self._latitude = check_latitude(latitude)

    def _get_latitude(self):
        return self._latitude

    longitude = property(_get_longitude, _set_longitude)
    latitude = property(_get_latitude, _set_latitude)

    def toLV95(self):
        wgstolv95 = Transformer.from_crs("EPSG:4326", "EPSG:2056")
        lv95 = wgstolv95.transform(self.longitude, self.latitude)
        return lv95

    def fromLV95(E,N):
        lv95towgs = Transformer.from_crs("EPSG:2056", "EPSG:4326")
        wgs84 = lv95towgs.transform(E, N)
        return wgs84

if __name__ == '__main__':
    P1 = WGS84Coord(47.5, 7.6)
    print(P1.longitude)          ----- Output -----> 47.5
    print(P1.latitude)          ----- Output -----> 7.599999999999994

    lv95 = P1.toLV95()
    print(lv95)                  ----- Output -----> (2612159.3926993245, 1261039.768937114)

    #----- LV95to WGS84

    E = lv95[0]
    N = lv95[1]

    print(WGS84Coord.fromLV95(E,N)) ----- Output -----> (47.500000010459885, 7.6000000074114915)
```

Aufgabe 2:

Gegeben sind folgende Objekte als WKT:

POINT(0 0)

LINESTRING(0 0,1 1,1 2)

POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))

MULTIPOINT(0 0,1 2)

MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))

MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)),((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))

GEOMETRYCOLLECTION(POINT(2 3),LINESTRING((2 3,3 4)))

- Berechnen Sie die Fläche der Objekte (wenn sinnvoll)
- Berechnen Sie die Länge der Objekte (wenn sinnvoll)
- Stellen Sie diese mit matplotlib dar

Hinweise:

- Dokumentation <https://shapely.readthedocs.io/en/stable/>
- Die Fläche lässt sich mit `object.area` berechnen
- Die Länge lässt sich mit `object.length` berechnen
- Den Geometry-Typ können Sie in Shapely mit `object.geom_type` abfragen.

```
from shapely.geometry import Point
import shapely.wkt
import matplotlib.pyplot as plt

# ----- Definition der Objekte -----
# angepasste Koordinaten, damit alle Objekte trotz Überlappung sichtbar
wkt1 = "POINT(0.5 3.5)"
point = shapely.wkt.loads(wkt1)

wkt2 = "LINESTRING(0 0,1.5 1.5,1 2)"
linestring = shapely.wkt.loads(wkt2)

wkt3 = "POLYGON((0 0,4.5 0,4.5 4.5,0 4.5,0 0),(1.5 1.5, 2 1.5, 2 2, 1.5 2,1.5 1.5))"
polygon = shapely.wkt.loads(wkt3)

wkt4 = "MULTIPOINT(0 0,1 2)"
multipoint = shapely.wkt.loads(wkt4)

wkt5 = "MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))"
multilinestring = shapely.wkt.loads(wkt5)

wkt6 = "MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)),((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))"
multipolygon = shapely.wkt.loads(wkt6)

wkt7 = "GEOMETRYCOLLECTION(POINT(2 3.5),LINESTRING(2 3,3 4))"
geometrycollection = shapely.wkt.loads(wkt7)

# ----- Fläche der Geometrien -----
f_polygon = polygon.area
print(f"Die Fläche des Polygon ist: {f_polygon}")

f_multipolygon = multipolygon.area
print(f"Die Fläche des Multipolygon ist : {f_multipolygon}")

# ----- Länge der Geometrien -----
l_linestring = linestring.length
print(f"Die Länge des Linestring ist: {l_linestring}")

l_polygon = polygon.length
print(f"Die Länge der Polygonseiten sind: {l_polygon}")

l_multipolygon = multipolygon.length
print(f"Die Länge der Multipolygonseiten sind: {l_multipolygon}")

l_multilinestring = multilinestring.length
print(f"Die Länge des Multilinestring ist: {l_multilinestring}")

l_geometrycollection = geometrycollection.length
print(f"Die Länge der Geometrycollection ist: {l_geometrycollection}")

# ----- Plotten der Geometrien -----
x1,y1 = point.xy
x2,y2 = linestring.xy
x3,y3 = polygon.exterior.xy

plt.plot(x1,y1, "ro")
plt.plot(x2,y2, "ko-")
plt.plot(x3,y3, "go-")

for geom4 in multipoint.geoms:
    plt.plot(*geom4.xy, "co-")

for geom5 in multilinestring.geoms:
    plt.plot(*geom5.xy, "mo-")

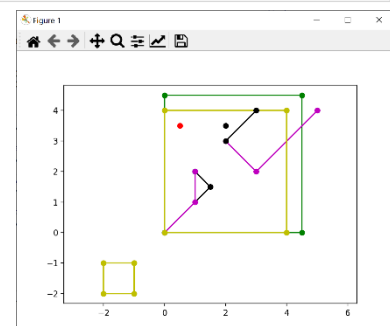
for geom6 in multipolygon.geoms:
    plt.plot(*geom6.exterior.xy, "yo-")

for geom7 in geometrycollection.geoms:
    plt.plot(*geom7.xy, "ko-")

plt.axis("equal")
plt.show()
```

Output:

```
Die Fläche des Polygon ist: 20.0
Die Fläche des Multipolygon ist : 16.0
Die Länge des Linestring ist: 2.82842712474619
Die Länge der Polygonseiten sind: 20.0
Die Länge der Multipolygonseiten sind: 24.0
Die Länge des Multilinestring ist: 6.656854249492381
Die Länge der Geometrycollection ist: 1.4142135623730951
```



Kapitel 11 - Folium & GeoPandas

Uebung

Aufgabe 1

Das Shapefile "daten/continent/continent.shp" enthält die Kontinente ["Africa", "Antarctica", "Asia", "Australia", "Europe", "North America", "Oceania", "South America"].

- Laden Sie das Shapefile mit geopandas und erstellen Sie das GeoDataFrame `gdfContinents`. Geben Sie dieses als Tabelle aus.
- In welchem Koordinatenreferenzsystem ist dieser Datensatz ?
- Erstellen Sie ein GeoDataFrame für Europa
- Erstellen Sie ein GeoDataFrame für alle Kontinente **ausser** Europa
- Plotten Sie die beiden GeoDataFrames in verschiedenen Farben

```
a)
import geopandas as gpd

import fiona
fiona.supported_drivers
{'AeronavFAA': 'r', 'ARCGEN': 'r', 'BNA': 'raw', 'DXF': 'raw', 'CSV': 'raw', 'OpenFileGDB': 'r', 'ESRIJSON': 'r', 'ESRI
Shapefile': 'raw', 'GeoJSON': 'rw', 'GeoJSONSeq': 'rw', 'GPKG': 'rw', 'GML': 'raw', 'GPX': 'raw', 'GPSTrackMaker': 'raw',
'Idrisi': 'r', 'MapInfo File': 'raw', 'DGN': 'raw', 'PCIDSK': 'r', 'S57': 'r', 'SEGY': 'r', 'SUA': 'r', 'TopoJSON': 'r'}
```

```
gdfContinents = gpd.read_file('daten/continent.shp', sep="\t", header=None, low_memory=False)
gdfContinents.head()
```

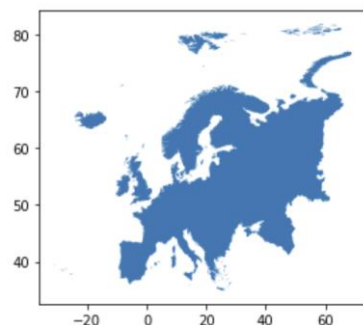
	geometry
0	MULTIPOLYGON (((93.27554 80.26361, 93.31304 80...
1	MULTIPOLYGON (((-25.28167 71.39166, -25.32889 ...
2	MULTIPOLYGON (((58.06138 81.68776, 57.98055 81...
3	MULTIPOLYGON (((0.69465 5.77337, 0.66667 5.803...
4	MULTIPOLYGON (((-81.71306 12.49028, -81.72014 ...

```
b)
gdfContinents.crs
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

```
c)
from shapely.geometry import Point
Europa = Point(7.638423, 47.534018)

gdfContinents.contains(Europa)

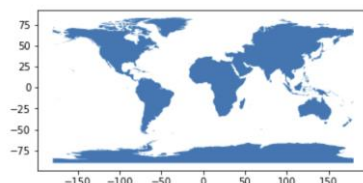
gdfContinents[gdfContinents.contains(Europa)].plot()
```



```
d)
from shapely.geometry import Point
ausserEuropa = Point(7.638423, 47.534018)

gdfContinents.contains(ausserEuropa)

gdfContinents[gdfContinents.contains(ausserEuropa)==False].plot()
```



Aufgabe 2

Die Erdbeben aus den Daten des USGS werden als GeoDataFrame geladen:

```
import requests

url = "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_week.geojson"

data = requests.get(url)
file = open("earthquakes.geojson", "wb")
file.write(data.content)
file.close()
eb = gpd.read_file("earthquakes.geojson")
```

a) Erstellen Sie unter Verwendung des zuvor erstellten GeoDataFrame "eb" folgende neue Geodataframes:

- eb3 enthält Erdbeben mit Magnitude [3.5, 4)
- eb4 enthält Erdbeben mit Magnitude [4, 5)
- eb5 enthält Erdbeben mit Magnitude 5 und grösser.

Hinweis: am einfachsten verwenden Sie dazu `eb.query("????")`

b) Bestimmen Sie die Anzahl Beben in eb3, eb4, eb5

c) Stellen Sie eb3, eb4 und eb5 auf einer Karte mit Folium dar (Erdbeben als Marker)

```
a)
import geopandas as gpd
import pandas as pd
import fiona

import requests

url = "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_week.geojson"

data = requests.get(url)
file = open("earthquakes.geojson", "wb")
file.write(data.content)
file.close()
eb = gpd.read_file("earthquakes.geojson")

eb3 = eb.query ("mag >= 3.5 and mag < 4")
eb4 = eb.query ("mag >= 4 and mag < 5")
eb5 = eb.query ("mag >= 5")

b)
eb3.shape[0]
44
eb4.shape[0]
187
eb5.shape[0]
54

c)
import folium
from folium.plugins import MarkerCluster

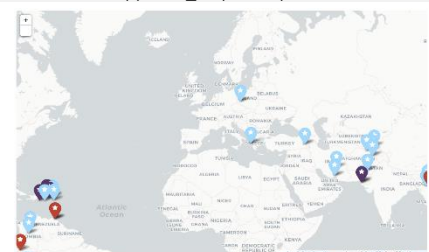
center = [47.534018, 7.638423]
karte = folium.Map(center, zoom_start=1.5, tiles='cartodbpositron')

for punkt in eb3.geometry:
    lat_eb3 = punkt.y
    lon_eb3 = punkt.x
    folium.Marker([lat_eb3, lon_eb3], icon=folium.Icon(color="darkpurple", prefix="fa", icon="star")).add_to(karte)

for punkt in eb4.geometry:
    lat_eb4 = punkt.y
    lon_eb4 = punkt.x
    folium.Marker([lat_eb4, lon_eb4], icon=folium.Icon(color="lightblue", prefix="fa", icon="star")).add_to(karte)

for punkt in eb5.geometry:
    lat_eb5 = punkt.y
    lon_eb5 = punkt.x
    folium.Marker([lat_eb5, lon_eb5], icon=folium.Icon(color="red", prefix="fa", icon="star")).add_to(karte)

karte
```



Kapitel 12 – Projektionen & Vektordaten, Teil 2: cartopy

Übung

Erstellen Sie ein Quiz welches einen Kanton zufällig auswählt und dessen Name geraten werden soll.

- a) Es sollen 4 Kantone als mögliche Auswahl präsentiert werden. Die Übung soll zunächst als Jupyter Notebook realisiert werden, analog zum Screenshot. Abgabe als Notebook File (.ipynb). Es soll nur ein Kanton ausgewählt und abgefragt werden.

```
import cartopy.crs as ccrs
from cartopy.feature import ShapelyFeature
import geopandas as gpd
import random
import matplotlib.pyplot as plt

kantone = gpd.read_file("daten/gemeindegrenzen/ggg_2021-LV95/shp/g1k21.shp", encoding="utf-8")
kantonegs84 = kantone.to_crs("EPSG:4326")
kantonegs84.to_file("daten/kantoneWGS84.shp", driver="Shapefile", encoding="utf-8")
kantone = gpd.read_file("daten/kantoneWGS84.shp", encoding="utf-8")

# Darstellung Kanton

kantone = gpd.read_file("daten/kantoneWGS84.shp", encoding="utf-8")

ktListe = list(range(1,27))
random.shuffle(ktListe)

ktnrList = list((ktListe[:4]))

gewahltektnr = ktnrList[0]
random.shuffle(ktnrList)

plt.figure(figsize=(15, 6))
ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)

kRandom = kantone.query(f"KTNR == {gewahltektnr}")

darstellung = kRandom.iloc[0][0]["geometry"]

bbox = darstellung.bounds

ax.set_extent([bbox[0],bbox[2],bbox[1],bbox[3]])

shape_feature = ShapelyFeature([darstellung],crs=ccrs.PlateCarree(), lw=1)
ax.add_feature(shape_feature)

plt.show()

# Auswahlliste

print("Welcher Kanton ist das?")

b = kantone.query(f"KTNR == {ktnrList[0]}")
c = kantone.query(f"KTNR == {ktnrList[1]}")
d = kantone.query(f"KTNR == {ktnrList[2]}")
e = kantone.query(f"KTNR == {ktnrList[3]}")

df = b.append([c,d,e])
print(df[["KTNAME", "KTNR"]])

# Eingabe Wert

val = input("Welches ist die die korrekte KTNR-Nr?: ")

v = int(val)

if int(val) == gewahltektnr:
    print("Die Antwort ist korrekt")
else:
    print("Die Antwort ist falsch")
```




```
# Uebung 12
# b) Im zweiten Schritt soll eine Applikation mit PyQt5 realisiert werden. Der Kanton kann als plot wie in Kapitel 9
# hinzugefügt werden. Als Auswahl sollen die Auswahlmöglichkeiten als Buttons integriert werden. Es sollen total 5 Kantone
# geraten werden und eine Maximalpunktzahl von 5 Punkten erreicht werden. Abgabe als .py File.

# -----

from PyQt5.QtWidgets import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgi as FigureCanvas
import matplotlib.pyplot as plt
from PyQt5.QtCore import *
import cartopy.crs as ccrs
from cartopy.feature import ShapelyFeature
import geopandas as gpd
import random

kantone = gpd.read_file("C:\\Users\\celin\\PR2\\uebung-12-celinaneumann\\daten\\kantoneWGS84.shp", encoding="utf-8")

class Window(QMainWindow):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Uebung 12b Kantonsquiz")

        # Initialisierung Punktezah und Zähler Anzahl beantwortete Fragen
        self.point = 0
        self.count = 0

        # Definition der Widgets
        layout = QVBoxLayout()
        figure = plt.figure(figsize=(16,9))

        form = QFormLayout()
        self.frage = QLabel("Welcher Kanton ist das?")
        self.start = QPushButton("Spiel starten")
        self.weiter = QPushButton("Weiter")
        self.auswahl0 = QPushButton(f"")
        self.auswahl1 = QPushButton(f"")
        self.auswahl2 = QPushButton(f"")
        self.auswahl3 = QPushButton(f"")
        self.rueckmeldung = QLabel(f"")
        self.spielende = QLabel(f"")

        form.addRow(self.frage)

        form_widget = QWidget()
        form_widget.setLayout(form)

        # Widgets erstellen
        self.canvas = FigureCanvas(figure)

        # Widgets dem Layout hinzufügen
        layout.addWidget(self.canvas)
        layout.addWidget(form_widget)
        layout.addWidget(self.start)
        layout.addWidget(self.auswahl0)
        layout.addWidget(self.auswahl1)
        layout.addWidget(self.auswahl2)
        layout.addWidget(self.auswahl3)
        layout.addWidget(self.rueckmeldung)
        layout.addWidget(self.spielende)
        layout.addWidget(self.weiter)

        # Connections
        self.start.clicked.connect(self.plot)
        self.weiter.clicked.connect(self.plot)
        self.auswahl0.clicked.connect(self.ausw0)
        self.auswahl1.clicked.connect(self.ausw1)
        self.auswahl2.clicked.connect(self.ausw2)
        self.auswahl3.clicked.connect(self.ausw3)

        # Hauptlayout setzten und anzeigen
        center = QWidget()
        center.setLayout(layout)
        self.setCentralWidget(center)
        self.show()

# -----

# Verarbeitung der Eingabeparameter und darstellung als Plot
def plot(self):
    plt.clf()
    self.rueckmeldung.setText(f"")
    self.spielende.setText(f"")

    L = list(range(0, len(kantone)))

    self.k0 = random.choice(L)
    self.k1 = random.choice(L)
    self.k2 = random.choice(L)
    self.k3 = random.choice(L)

    a = [self.k0, self.k1, self.k2, self.k3]
    self.k = random.choice(a)
```

```

# Komponenten zur Darstellung der Kantonsgeometrie
ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)
geometry = kantone.iloc(0)[self.k]["geometry"]
bbox = geometry.bounds
ax.set_extent([bbox[0],bbox[2],bbox[1],bbox[3]]) #Boudbndingbox aufrund Ausdehnung Kanton
shape_feature = ShapelyFeature([geometry], crs=ccrs.PlateCarree(), fc="blue", ec='black', lw=1)
ax.add_feature(shape_feature)

# Kantonsnamen für Fragestellung bestimmen und auf Buttons schreiben
self.a0 = kantone.iloc(0)[self.k0]["KTNAME"]
self.a1 = kantone.iloc(0)[self.k1]["KTNAME"]
self.a2 = kantone.iloc(0)[self.k2]["KTNAME"]
self.a3 = kantone.iloc(0)[self.k3]["KTNAME"]

self.answahl0.setText(f"{self.a0}")
self.answahl1.setText(f"{self.a1}")
self.answahl2.setText(f"{self.a2}")
self.answahl3.setText(f"{self.a3}")

plt.axis("equal")
self.canvas.draw()

# -----

# Funktionen der Kantonsbuttons definieren
def ausw0(self):
    if self.k == self.k0:
        self.rueckmeldung.setText(f"{self.a0} ist richtig :)")
        self.count = self.count+1
        self.point = self.point+1
    else:
        self.rueckmeldung.setText(f"{self.a0} ist leider falsch :(")
        self.count = self.count+1

    if self.count == 5:
        self.spielende.setText(f"Das Spiel ist vorbei Sie haben {self.point} Punkt(e) von maximal 5 Punkten geholt.
Starten Sie das Spiel erneut mit 'Spiel starten'.")
        self.point = 0
        self.count = 0

def ausw1(self):
    if self.k == self.k1:
        self.rueckmeldung.setText(f"{self.a1} ist richtig :)")
        self.count = self.count+1
        self.point = self.point+1
    else:
        self.rueckmeldung.setText(f"{self.a1} ist leider falsch :(")
        self.count = self.count+1

    if self.count == 5:
        self.spielende.setText(f"Das Spiel ist vorbei Sie haben {self.point} Punkt(e) von maximal 5 Punkten geholt.
Starten Sie das Spiel erneut mit 'Spiel starten'.")
        self.point = 0
        self.count = 0

def ausw2(self):
    if self.k == self.k2:
        self.rueckmeldung.setText(f"{self.a2} ist richtig :)")
        self.count = self.count+1
        self.point = self.point+1
    else:
        self.rueckmeldung.setText(f"{self.a2} ist leider falsch :(")
        self.count = self.count+1

    if self.count == 5:
        self.spielende.setText(f"Das Spiel ist vorbei Sie haben {self.point} Punkt(e) von maximal 5 Punkten geholt.
Starten Sie das Spiel erneut mit 'Spiel starten'.")
        self.point = 0
        self.count = 0

def ausw3(self):
    if self.k == self.k3:
        self.rueckmeldung.setText(f"{self.a3} ist richtig :)")
        self.count = self.count+1
        self.point = self.point+1
    else:
        self.rueckmeldung.setText(f"{self.a3} ist leider falsch :(")
        self.count = self.count+1

    if self.count == 5:
        self.spielende.setText(f"Das Spiel ist vorbei. Sie haben {self.point} Punkt(e) von maximal 5 Punkten geholt.
Starten Sie das Spiel erneut mit 'Spiel starten'.")
        self.point = 0
        self.count = 0

# -----

app = QApplication([])
fenster = Window()
app.exec()

```