Modulabschlussprüfung Programmierung II

Repertorium

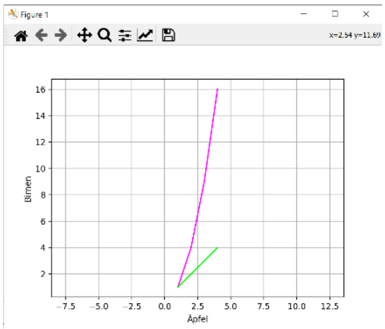
Kapitel 1 – Einführung git & GitHub

Kapitel 2 - Numerisches Python I

Vorlesung:

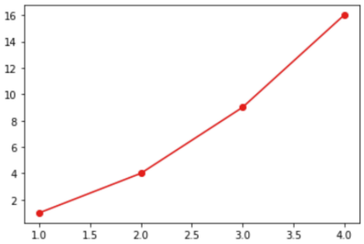
- **01_example.py**

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], color="#FF00FF")
plt.plot([1,2,3,4], [1,2,3,4], color="#00FF00")
plt.xlabel("Äpfel")
plt.ylabel("Birnen")
plt.axis("equal")
plt.grid(True)
plt.show()
```



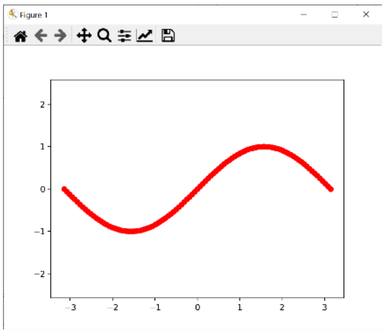
- **02_plotten.ipynb**

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], "ro-");
p
```



- **03_sinuskurve.py**

```
import matplotlib.pyplot as plt
import math

s = 2*math.pi
n = 100
teilstück = s / n
xlist = []
ylist = []

for i in range(0,n+1):
    x = -math.pi + i*teilstück
    xlist.append(x)
    y = math.sin(x)
    ylist.append(y)

plt.plot(xlist,ylist, "ro--")
plt.axis("equal")
plt.show()
```



- **04_numerik.py**

```
import numpy as np

a = np.array([1,2,3,4], dtype=np.float64)
b = np.array(list(range(0,11))) / 10

c = np.array([ [1,2,3],[4,5,6],[7,8,9] ])

print(c[:,0])

print(c.shape)

print(10 in c)
```

```
Output:
[1 4 7]

(3,3)

False
```

- **05_arrays.ipynb**

```
import numpy as np
```

```
np.zeros([3,4])
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
a = np.zeros([3,3])
a[2,2] = 5
a
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 5.]])
```

```
a = np.arange(0,10.5,0.5)
a
array([ 0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ,
       5.5, 6. , 6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5, 10. ])
```

```
a = np.linspace(0,1, 11)
a
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]
```

```
np.pi * a
array([0. , 0.31415927, 0.62831853, 0.9424778 , 1.25663706,
       1.57079633, 1.88495559, 2.19911486, 2.51327412, 2.82743339,
       3.14159265])
```

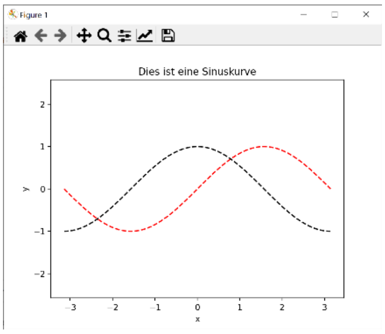
```
np.random.random(10)*100
array([85.02796136, 57.83834166, 87.29521745, 49.53857336, 21.20886912,
       73.30682197, 68.52896197, 47.9098261 , 67.81530219, 7.82074031])
```

```
np.ones([8,7])
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]]
```

```
a = np.arange(0,10)
a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

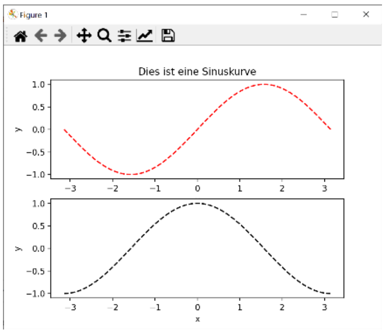
- **06_numpyplot.py**

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

x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)
y2 = np.cos(x)

plt.plot(x,y,"r--")
plt.plot(x,y2,"k--")
plt.axis("equal")
plt.title("Dies ist eine Sinuskurve")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



- **07_subplot.py**

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

x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)
y2 = np.cos(x)

plt.subplot(2,1,1)
plt.title("Dies ist eine Sinuskurve")
plt.plot(x,y,"r--")
plt.xlabel("x")
plt.ylabel("y")

plt.subplot(2,1,2)
plt.plot(x,y2,"k--")
plt.xlabel("x")
plt.ylabel("y")

plt.show()
```



- **Eigene_Python_module_importieren.ipynb**

Die python Datei numerik.py mit folgenden Inhalt:

```
import numpy as np

a = np.array([1,2,3,4], dtype=np.float64)
b = np.array(list(range(0,11))) / 10

c = np.array([ [1,2,3],[4,5,6],[7,8,9] ])

new_var = np.random.random(10)
```

lässt sich wie ein normales python Modul importieren (dafür muss die Datei in dem gleichen Ordner gespeichert sein, sonst müsste man die sys.path Variable anpassen...).

```
import numerik
from numerik import new_var
```

- **Hilfe_Funktionen.ipynb**

help(), func? und func??

```
# was macht meshgrid überhaupt?
help(np.meshgrid)

# das gleiche wie oben
np.meshgrid?
```

```
# docstring + source code
np.meshgrid??
```

Kapitel 3 – Objektorientierung, Teil 1

Vorlesung

- **01_beispiel.py**

punkt.py

```
# Klassendefinition
class Punkt:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def ausgabe(self, untereinander):
        if untereinander:
            print("x=", self.x)
            print("y=", self.y)
        else:
            print(f"Punkt: [{self.x},{self.y}]")

    def distanz(self, b):
        return ((self.x-b.x)**2 + (self.y-b.y)**2)**0.5

if __name__ == "__main__":
    a = Punkt(3,4)
    b = Punkt(2,4)
    a.ausgabe(False)
    b.ausgabe(False)
    print(a.distanz(b))
```

```
import punkt

a = punkt.Punkt() # Instanz
b = punkt.Punkt(1.2,1.1) # Instanz

a.ausgabe(True)
b.ausgabe(False)

d1 = a.distanz(b)
d2 = b.distanz(a)
d3 = a.distanz(a)

print(d1)
print(d2)
print(d3)
```

```
Output:
x= 0
y= 0
Punkt: [1.2,1.1]


1.6278820596099708
1.6278820596099708
0.0
```

- **02_temperatur.py**

```
class Temperatur:
    def __init__(self, celsius=0):
        self.setValue(celsius)

    def setValue(self, v):
        if v < -273.15:
            print("Warnung: Temperatur-Wert wurde korrigiert auf -273.15 Celsius")
            v = -273.15
            #raise ValueError("Absoluter Nullpunkt ist -273.15 Celsius")
        self._value = v

    def getValue(self):
        return self._value

    def setValueF(self, f):
        self.setValue((f-32) / 1.8)

    def getValueF(self):
        c = self.getValue()
        f = c * 1.8 + 32
        return f

    celsius = property(getValue, setValue)
    fahrenheit = property(getValueF, setValueF)

## ----------------------------------------------------------

t0 = Temperatur(20)
print(t0.celsius)
print(t0.fahrenheit)

t1 = Temperatur()
t1.fahrenheit = 100
print(t1.celsius)
t1.fahrenheit = -1000
```

```
Output:

20
68.0


37.77777777777778
Warnung: Temperatur-Wert wurde korrigiert auf -273.15 Celsius
```

Kapitel 4 – Objektorientierung, Teil 2

Vorlesung

- **temperatur.py**

```python
class Temperatur:
    def __init__(self, c):
        self.celsius = c

    def __str__(self):
        return str(self.celsius) + chr(8451)

    def __gt__(self, other):
        return self.celsius > other.celsius

    def __lt__(self, other):
        return self.celsius < other.celsius

    def __eq__(self, other):
        return self.celsius == other.celsius

t_muttenz = Temperatur(9.0)
t_zurich = Temperatur(11)

print(t_muttenz)

if t_zurich > t_muttenz:
    print("😠😠😠😠😠😠😠😠")
else:
    print("HAHAHA")

if t_zurich == t_muttenz:
    print("ok")
```

- **tempservice.py**

```
apikey.py


# rename this file to apikey.py
# api_key = "<INSERT API-KEY HERE>"
```

```python
import requests
import apikey

class Temperatur:
    def __init__(self, city):
        url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={apikey.api_key}&units=metric"
        data = requests.get(url)
        x = data.json()["main"]
        self.celsius = x["temp"]
        self.city = city

    def __str__(self):
        return f"Temperatur in {self.city} ist {self.celsius} {chr(8451)}"

    def __gt__(self, other):
        return self.celsius > other.celsius

    def __ge__(self, other):
        return self.celsius >= other.celsius

    def __lt__(self, other):
        return self.celsius < other.celsius

    def __le__(self, other):
        return self.celsius <= other.celsius

    def __eq__(self, other):
        return self.celsius == other.celsius

    def __ne__(self, other):
        return self.celsius != other.celsius


t_zurich = Temperatur("Zürich,Switzerland")
t_basel = Temperatur("Basel,Switzerland")

print(t_zurich)
print(t_basel)

if t_zurich > t_basel:
    print("oh, nein!!!!!! ARGH!!!!")
else:
    print("Juhuuuu!!!!")
```

- **vector2.py**

```
class Vector2:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2(self.x+other.x, self.y+other.y)

    def __sub__(self, other):
        return Vector2(self.x-other.x, self.y-other.y)

    def __mul__(self, other):
        if type(other) == int or type(other) == float:
            return Vector2(self.x * other, self.y * other)
        else:
            return Vector2(self.x*other.x, self.y*other.y)

    def __rmul__(self, other):
        if type(other) == int or type(other) == float:
            return Vector2(self.x * other, self.y * other)
        else:
            return Vector2(self.x*other.x, self.y*other.y)

    def __neg__(self):
        return Vector2(-self.x, -self.y)

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

#------------------------------------------------------------------------------

a = Vector2(3,4)
b = Vector2(2,3)
c = Vector2(4,5)

resultat = ( a + b + c ) + a + b + a + b + ( c + a ) + a + a
print(resultat)      ------------- Output ------------------>  VECTOR (32 43)

a = Vector2(4,5)
b = Vector2(2,3)
c = a * b
print(c)             ------------- Output ------------------>  VECTOR (8 15)

d = a * 2
print(d)             ------------- Output ------------------>  VECTOR (8 10)

d = 2 * a
print(d)             ------------- Output ------------------>  VECTOR (8 10)

a = Vector2(1,2)
b = Vector2(2,3)
c = Vector2(2,2)

d = a + (b * c)
print(d)             ------------- Output ------------------>  VECTOR (5 8)

a = Vector2(1,2)
a += Vector2(1,1)
print(a)             ------------- Output ------------------>  VECTOR (2 3)

b = -a
print(b)             ------------- Output ------------------>  VECTOR (-2 -3)
```

Kapitel 5 – Objektorientierung, Teil 3 Theorie

Vorlesung

- **01_HatBeziehung.py**

```python
class Studiengang:
    def __init__(self, hochschule, name):
        self.hochschule = hochschule
        self.name = name

## -------------------------------------------

class Kurs:
    def __init__(self, name):
        self.name = name
        self.students = []

    def addStudent(self, student):
        self.students.append(student)

    def __str__(self):
        s = "KURS: " + self.name + "\n"
        for student in self.students:
            s += student.vorname + " " + student.nachname + " " + student.studiengang.name + "\n"
        return s
## -------------------------------------------

class Student:
    def __init__(self, vorname, nachname, geschlecht, studiengang):
        self.matrikelnummer = ""
        self.vorname = vorname
        self.nachname = nachname
        self.adresse = ""
        self.studiengang = studiengang
        self.geschlecht = geschlecht


geomatik = Studiengang("HABG", "Geomatik")
architektur = Studiengang("HABG", "Architektur")

student1 = Student("Hans", "Meier", "m", geomatik)
student2 = Student("Alexandra", "Müller", "w", geomatik)
student3 = Student("Joachim", "Huber", "m", architektur)

student3.studiengang = geomatik

prog = Kurs("Programmieren")
prog.addStudent(student1)
prog.addStudent(student2)
prog.addStudent(student3)

dbv = Kurs("DBV")
dbv.addStudent(student1)
dbv.addStudent(student3)

print(prog)
print(dbv)

student1.vorname = "Franz"

print(prog)
```

```
Output:

KURS: Programmieren
Hans Meier Geomatik
Alexandra Müller Geomatik
Joachim Huber Geomatik

KURS: DBV
Hans Meier Geomatik
Joachim Huber Geomatik

KURS: Programmieren
Franz Meier Geomatik
Alexandra Müller Geomatik
Joachim Huber Geomatik
```

- **02_Vererbung.py**

```
class Fahrzeug:
    def __init__(self, farbe, räder):
        self.farbe = farbe
        self.räder = räder
        self.fahrgestellnummer = ""
        self.sitzplätze = 0

    def fahren(self):
        print("fährt weg...")

## --------------------------------------------

class PKW(Fahrzeug):
    def __init__(self, schiebedach, farbe, räder):
        super().__init__(farbe, räder)
        self.schiebedach = schiebedach

    def __str__(self):
        return f"**PKW** Schiebedach: {self.schiebedach}, Farbe {self.farbe}, Räder: {self.räder}"

##--------------------------------------------

class Fahrrad(Fahrzeug):
    def __init__(self, rahmengrösse, farbe):
        super().__init__(farbe, 2)
        self.rahmengrösse = rahmengrösse

    def __str__(self):
        return f"**FAHRRAD** Rahmengrösse: {self.rahmengrösse}, Farbe {self.farbe}"

##----------------------------------------------

tesla = PKW(True, "grün", 4)
print(tesla)       ------------- Output ------------------>    **PKW** Schiebedach: True, Farbe grün, Räder: 4
tesla.fahren()     ------------- Output ------------------>    fährt weg...

fiat = PKW(False, "schwarz", 3)
print(fiat)        ------------- Output ------------------>    **PKW** Schiebedach: False, Farbe schwarz, Räder: 3

scott = Fahrrad(54, "grau")
print(scott)       ------------- Output ------------------>    **FAHRRAD** Rahmengrösse: 54, Farbe grau
```

- **figur.py**

```
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")
```

```
from punkt import Punkt
import math

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

    def umfang(self):
        return 0

    def __str__(self):
        return self.name

#--------------------------------------------

class Kreis(Figur):
    def __init__(self, M=Punkt(0,0), r=1):
        super().__init__("Kreis")
        if type(M) != Punkt:
            raise TypeError("M muss Klasse Punkt sein")

        self.Mittelpunkt = M
        self.radius = r

    def umfang(self):
        return 2*self.r*math.pi

    def __str__(self):
        return f"{self.name}: Mittelpunkt: {self.Mittelpunkt}, Radius: {self.radius}"


k1 = Kreis()
k2 = Kreis(Punkt(1,1), 4)

print(k1)         ------------- Output ------------------>    Kreis: Mittelpunkt: Punkt(0,0), Radius: 1
print(k2)         ------------- Output ------------------>    Kreis: Mittelpunkt: Punkt(1,1), Radius: 4
```

Kapitel 6 – GUI Programmierung, Teil 1

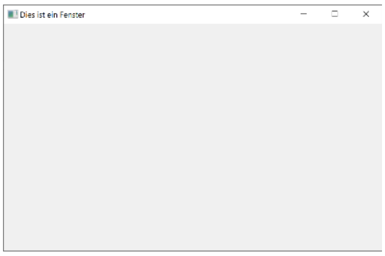
Vorlesung

- **_template.py**

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Grid Layout")

        # layout = QGridLayout()

        # Widgets erstellen

        # Widgets dem Layout hinzufügen

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()


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

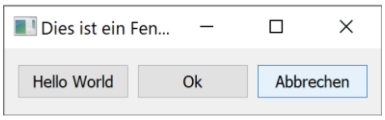
- **00_window.py**

```
from PyQt5.QtWidgets import *

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

        #self.setGeometry(20,120,640,480)
        self.setMinimumWidth(1280)
        self.setMinimumHeight(768)
        self.setWindowTitle("Dies ist ein Fenster")
        self.show()


app = QApplication([])
fenster = Fenster()
fenster.raise_()
app.exec()
```



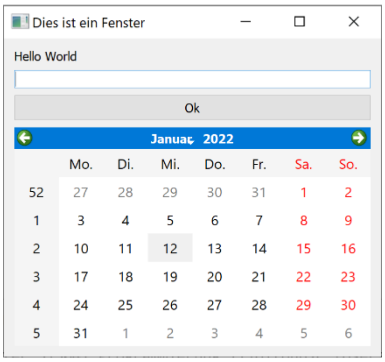
- **01_hboxlayout.py**

```
from PyQt5.QtWidgets import *

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

        #self.setMinimumWidth(1280)
        #self.setMinimumHeight(768)
        self.setWindowTitle("Dies ist ein Fenster")

        layout = QHBoxLayout()

        button1 = QPushButton("Hello World")
        button2 = QPushButton("Ok")
        button3 = QPushButton("Abbrechen")

        layout.addWidget(button1)
        layout.addWidget(button2)
        layout.addWidget(button3)

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


app = QApplication([])
fenster = Fenster()
fenster.raise_()
app.exec()
```



- **02_widgets.py**

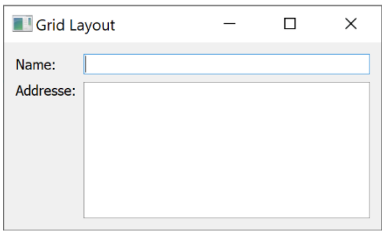
```
from PyQt5.QtWidgets import *

class Fenster(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dies ist ein Fenster")

        layout = QVBoxLayout()

        label = QLabel("Hello World")
        edit = QLineEdit()
        button = QPushButton("Ok")
        calendar = QCalendarWidget()

        layout.addWidget(label)
        layout.addWidget(edit)
        layout.addWidget(button)
        layout.addWidget(calendar)

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


app = QApplication([])
fenster = Fenster()
fenster.raise_()
app.exec()
```



	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.
52	27	28	29	30	31	1	2
1	3	4	5	6	7	8	9
2	10	11	12	13	14	15	16
3	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30
5	31	1	2	3	4	5	6

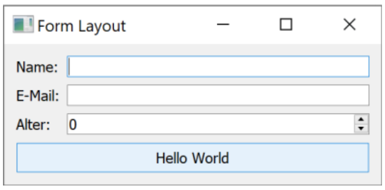
- **03_gridlayout.py**

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Grid Layout")

        layout = QGridLayout()

        nameLabel = QLabel("Name:")
        nameLine = QLineEdit()
        addressLabel = QLabel("Addresse:")
        addressLine = QTextEdit()

        layout.addWidget(nameLabel, 0, 0)
        layout.addWidget(nameLine, 0, 1)
        layout.addWidget(addressLabel, 1, 0, Qt.AlignTop)
        layout.addWidget(addressLine, 1, 1)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()


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



- **04_formlayout.py**

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Grid Layout")

        layout = QFormLayout()

        # Widgets erstellen
        name = QLineEdit()
        email = QLineEdit()
        alter = QSpinBox()
        button = QPushButton("Hello World")

        # Widgets dem Layout hinzufügen
        layout.addRow("Name:", name)
        layout.addRow("E-Mail:", email)
        layout.addRow("Alter:", alter)
        layout.addRow(button)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()


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



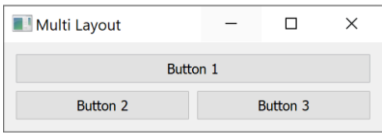
Name:

E-Mail:

Alter: 0

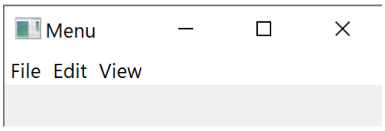
- **05_multilayout.py**

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Grid Layout")

        layout_start = QVBoxLayout()
        layout2 = QHBoxLayout()

        # Widgets erstellen
        button1 = QPushButton("Button 1")
        button2 = QPushButton("Button 2")
        button3 = QPushButton("Button 3")

        # Widgets dem Layout hinzufügen
        layout_start.addWidget(button1)
        layout2.addWidget(button2)
        layout2.addWidget(button3)

        layout_start.addLayout(layout2)

        center = QWidget()
        center.setLayout(layout_start)

        self.setCentralWidget(center)
        self.show()


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



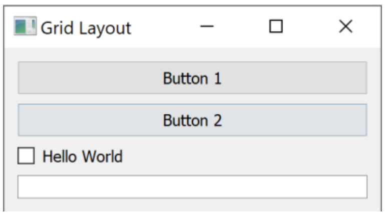
- **06_menu.py**

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Menu")

        menubar = self.menuBar()

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

        open = QAction("Öffnen", self)
        save = QAction("Speichern", self)
        quit = QAction("Exit", self)

        quit.setMenuRole(QAction.QuitRole)

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

        open.triggered.connect(self.doOpen)
        quit.triggered.connect(self.doQuit)

        self.show()

    def doOpen(self):
        print("Datei öffnen!!!")

    def doQuit(self):
        exit(0)

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



- **07_signal_slot.py**

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Grid Layout")

        layout = QVBoxLayout()

        # Widgets erstellen

        button1 = QPushButton("Button 1")
        button2 = QPushButton("Button 2")
        checkbox = QCheckBox("Hello World")
        self.name = QLineEdit()


        # Widgets dem Layout hinzufügen

        layout.addWidget(button1)
        layout.addWidget(button2)
        layout.addWidget(checkbox)
        layout.addWidget(self.name)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

        ########

        button1.clicked.connect(self.Knopf1)
        button2.clicked.connect(self.Knopf2)
        checkbox.stateChanged.connect(self.MyCheckBox)

    def MyCheckBox(self, state):
        if state == Qt.CheckState.Checked:
            print("Checkbox ist gewählt!")
        elif state == Qt.CheckState.Unchecked:
            print("Checkbox ist nicht gewählt")

    def Knopf1(self):
        print("Line Edit hat den Wert: " + self.name.text())

    def Knopf2(self):
        self.name.setText("Hello World")


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

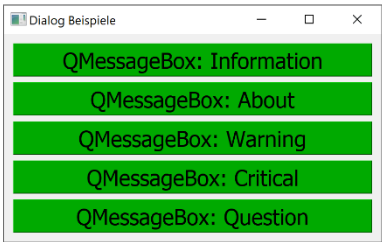


Kapitel 7 – GUI Programmierung, Teil 2

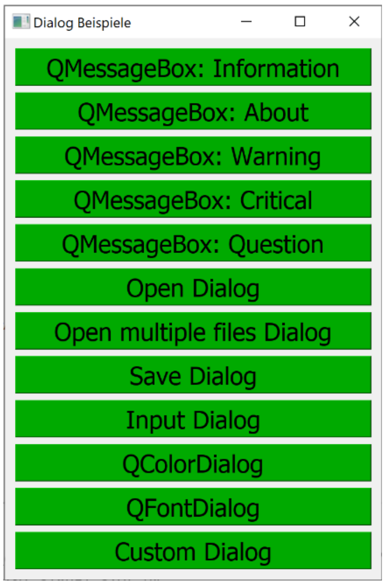
Vorlesung

- **dialog.py**

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *


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

        self.setWindowTitle("Dialog Beispiele")
        layout = QVBoxLayout()

        button1 = QPushButton("QMessageBox: Information")
        button2 = QPushButton("QMessageBox: About")
        button3 = QPushButton("QMessageBox: Warning")
        button4 = QPushButton("QMessageBox: Critical")
        button5 = QPushButton("QMessageBox: Question")

        button1.clicked.connect(self.button1_clicked)
        button2.clicked.connect(self.button2_clicked)
        button3.clicked.connect(self.button3_clicked)
        button4.clicked.connect(self.button4_clicked)
        button5.clicked.connect(self.button5_clicked)


        style = """QPushButton { font-size: 48px; background-color: #00AA00; }
                   QPushButton:pressed {font-size: 48px; background-color: #AA0000}"""
        button1.setStyleSheet(style)
        button2.setStyleSheet(style)
        button3.setStyleSheet(style)
        button4.setStyleSheet(style)
        button5.setStyleSheet(style)

        layout.addWidget(button1)
        layout.addWidget(button2)
        layout.addWidget(button3)
        layout.addWidget(button4)
        layout.addWidget(button5)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)

        self.show()

    def button1_clicked(self):
        QMessageBox.information(self, "Titel", "<h1>Hello World</h1>Python macht Spass<br/>Dies ist eine Zeile")

    def button2_clicked(self):
        QMessageBox.about(self, "Titel", "Dieses Programm wurde mit PyQT5 erstellt")

    def button3_clicked(self):
        QMessageBox.warning(self, "Titel", "Disk ist voll, das File konnte nicht geschrieben werden")

    def button4_clicked(self):
        QMessageBox.critical(self, "Stop", "Das Konfigurations-File konnte nicht geladen werden. Das Programm muss
beendet werden.")
        self.close()

    def button5_clicked(self):
        antwort = QMessageBox.question(self, "Frage", "Ist Python eine gute Spache?", QMessageBox.Yes, QMessageBox.No)

        if antwort == QMessageBox.Yes:
            QMessageBox.information(self, "Python", "Ja, das ist klar")
        else:
            QMessageBox.critical(self, "Buuuuuuh!!!!", "Ok, das Programm wird beendet")
            self.close()

app = QApplication([])
f = Fenster()
app.exec()
```

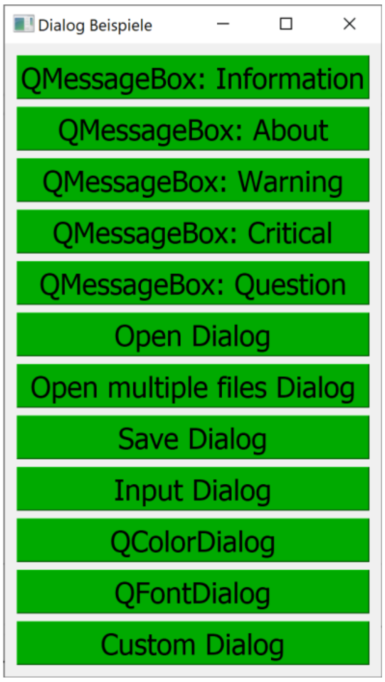


- **dialog2.py**



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

class Dialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        label = QLabel("Dies ist ein Label")
        button = QPushButton("Ok")
        layout = QVBoxLayout()

        layout.addWidget(label)
        layout.addWidget(button)
        self.setLayout(layout)
        button.clicked.connect(self.button_clicked)

    def button_clicked(self):
        self.close()

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

        self.setWindowTitle("Dialog Beispiele")
        layout = QVBoxLayout()

        buttons = []

        buttons.append(QPushButton("QMessageBox: Information"))
        buttons.append(QPushButton("QMessageBox: About"))
        buttons.append(QPushButton("QMessageBox: Warning"))
        buttons.append(QPushButton("QMessageBox: Critical"))
        buttons.append(QPushButton("QMessageBox: Question"))
        buttons.append(QPushButton("Open Dialog"))
        buttons.append(QPushButton("Open multiple files Dialog"))
        buttons.append(QPushButton("Save Dialog"))
        buttons.append(QPushButton("Input Dialog"))
        buttons.append(QPushButton("QColorDialog"))
        buttons.append(QPushButton("QFontDialog"))
        buttons.append(QPushButton("Custom Dialog"))


        buttons[0].clicked.connect(self.button1_clicked)
        buttons[1].clicked.connect(self.button2_clicked)
        buttons[2].clicked.connect(self.button3_clicked)
        buttons[3].clicked.connect(self.button4_clicked)
        buttons[4].clicked.connect(self.button5_clicked)
        buttons[5].clicked.connect(self.button6_clicked)
        buttons[6].clicked.connect(self.button7_clicked)
        buttons[7].clicked.connect(self.button8_clicked)
        buttons[8].clicked.connect(self.button9_clicked)
        buttons[9].clicked.connect(self.button10_clicked)
        buttons[10].clicked.connect(self.button11_clicked)
        buttons[11].clicked.connect(self.button12_clicked)

        style = """QPushButton { font-size: 48px; background-color: #00AA00; }
                   QPushButton:pressed {font-size: 48px; background-color: #AA0000}"""

        for button in buttons:
            button.setStyleSheet(style)
            layout.addWidget(button)


        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)

        self.show()

    def button1_clicked(self):
        QMessageBox.information(self, "Titel", "<h1>Hello World</h1>Python macht Spass<br/>Dies ist eine Zeile")

    def button2_clicked(self):
        QMessageBox.about(self, "Titel", "Dieses Programm wurde mit PyQT5 erstellt")

    def button3_clicked(self):
        QMessageBox.warning(self, "Titel", "Disk ist voll, das File konnte nicht geschrieben werden")

    def button4_clicked(self):
        QMessageBox.critical(self, "Stop", "Das Konfigurations-File konnte nicht geladen werden. Das Programm muss
beendet werden.")
        self.close()

    def button5_clicked(self):
        antwort = QMessageBox.question(self, "Frage", "Ist Python eine gute Spache?", QMessageBox.Yes, QMessageBox.No)

        if antwort == QMessageBox.Yes:
            QMessageBox.information(self, "Python", "Ja, das ist klar")
        else:
            QMessageBox.critical(self, "Buuuuuuh!!!!", "Ok, das Programm wird beendet")
            self.close()

    def button6_clicked(self):
        dateifilter = "Textdatei (*.txt *.ttt);;Python File (*.py)"

        path = QStandardPaths.standardLocations(QStandardPaths.DesktopLocation)[0]

        filename, filter = QFileDialog.getOpenFileName(self, "Datei öffnen", path, dateifilter)

        if filename != "":
            QMessageBox.information(self, "File", f"<h1>{filename}</h1><h2>{filter}</h2>")
        else:
            QMessageBox.warning(self, "Kein File", "Es wurde kein File ausgewählt")


    def button7_clicked(self):
        filenamen, filter = QFileDialog.getOpenFileNames(self, "Dateien öffnen", "", "Text (*.txt)")
        print(filenamen)

    def button8_clicked(self):
        filename, filter = QFileDialog.getSaveFileName(self, "Speichern", "", "Python (*.py)")
        print(filename, filter)

    def button9_clicked(self):
        wert, ok = QInputDialog.getItem(self, "Auswahl", "Welches Land ist schöner ?", ["Schweiz", "Deutschland",
"Österreich"], 1, True)

        wert, ok = QInputDialog.getDouble(self, "Titel", "Text")

        wert, ok = QInputDialog.getInt(self, "Titel", "Text", 20, 10, 30)
        if ok:
            print(wert)

    def button10_clicked(self):
        farbe = QColorDialog.getColor(initial=QColor(0,0,255))
        print(farbe.red(), farbe.green(), farbe.blue())

    def button11_clicked(self):
        font = QFontDialog.getFont()

    def button12_clicked(self):
        d = Dialog(self)
        d.exec()

app = QApplication([])
f = Fenster()
app.exec()
```

- **dialog3.py**



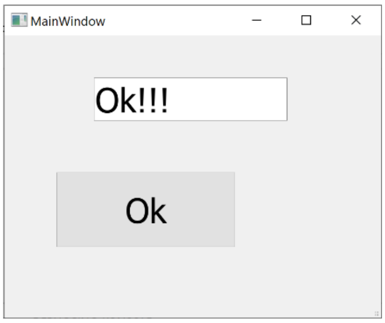
```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *

style = """* {font-size: 48px;}
                    QPushButton { font-size: 48px; background-color: #00AA00; }
                   QPushButton:pressed {font-size: 48px; background-color:
#AA0000}"""

class Dialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        label = QLabel("Dies ist ein Label")
        button = QPushButton("Ok")
        layout = QVBoxLayout()

        layout.addWidget(label)
        layout.addWidget(button)
        self.setLayout(layout)
        self.setStyleSheet(style)
        button.clicked.connect(self.button_clicked)

    def button_clicked(self):
        self.close()
#---------------------------------------------------------------------------------
------

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

        self.setWindowTitle("Dialog Beispiele")
        layout = QVBoxLayout()

        buttons = []

        buttons.append(QPushButton("QMessageBox: Information"))
        buttons.append(QPushButton("QMessageBox: About"))
        buttons.append(QPushButton("QMessageBox: Warning"))
        buttons.append(QPushButton("QMessageBox: Critical"))
        buttons.append(QPushButton("QMessageBox: Question"))
        buttons.append(QPushButton("Open Dialog"))
        buttons.append(QPushButton("Open multiple files Dialog"))
        buttons.append(QPushButton("Save Dialog"))
        buttons.append(QPushButton("Input Dialog"))
        buttons.append(QPushButton("QColorDialog"))
        buttons.append(QPushButton("QFontDialog"))
        buttons.append(QPushButton("Custom Dialog"))

        # Mit getattr kann die Mathode in der Klasse direkt gesucht werden (Advanced)
        # Man könnte sich auch die untere for-Schleife sparen und alles in eine for-Schleife packen
        for i in range(0,len(buttons)):
            function = getattr(self,f"button{i+1}_clicked")
            buttons[i].clicked.connect(function)

        for button in buttons:
            button.setStyleSheet(style)
            layout.addWidget(button)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)

        self.show()

    def button1_clicked(self):
        QMessageBox.information(self, "Titel", "<h1>Hello World</h1>Python macht Spass<br/>Dies ist eine Zeile")

    def button2_clicked(self):
        QMessageBox.about(self, "Titel", "Dieses Programm wurde mit PyQT5 erstellt")

    def button3_clicked(self):
        QMessageBox.warning(self, "Titel", "Disk ist voll, das File konnte nicht geschrieben werden")

    def button4_clicked(self):
        QMessageBox.critical(self, "Stop", "Das Konfigurations-File konnte nicht geladen werden. Das Programm muss
beendet werden.")
        self.close()

    def button5_clicked(self):
        antwort = QMessageBox.question(self, "Frage", "Ist Python eine gute Spache?", QMessageBox.Yes, QMessageBox.No)

        if antwort == QMessageBox.Yes:
            QMessageBox.information(self, "Python", "Ja, das ist klar")
        else:
            QMessageBox.critical(self, "Buuuuuuh!!!!", "Ok, das Programm wird beendet")
            self.close()
    def button6_clicked(self):
        dateifilter = "Textdatei (*.txt *.ttt);;Python File (*.py)"

        path = QStandardPaths.standardLocations(QStandardPaths.DesktopLocation)[0]

        filename, filter = QFileDialog.getOpenFileName(self, "Datei öffnen", path, dateifilter)

        if filename != "":
            QMessageBox.information(self, "File", f"<h1>{filename}</h1><h2>{filter}</h2>")
        else:
            QMessageBox.warning(self, "Kein File", "Es wurde kein File ausgewählt")


    def button7_clicked(self):
        filenamen, filter = QFileDialog.getOpenFileNames(self, "Dateien öffnen", "", "Text (*.txt)")
        print(filenamen)

    def button8_clicked(self):
        filename, filter = QFileDialog.getSaveFileName(self, "Speichern", "", "Python (*.py)")
        print(filename, filter)

    def button9_clicked(self):
        wert, ok = QInputDialog.getItem(self, "Auswahl", "Welches Land ist schöner ?", ["Schweiz", "Deutschland",
"Österreich"], 1, True)

        wert, ok = QInputDialog.getDouble(self, "Titel", "Text")

        wert, ok = QInputDialog.getInt(self, "Titel", "Text", 20, 10, 30)
        if ok:
            print(wert)

    def button10_clicked(self):
        farbe = QColorDialog.getColor(initial=QColor(0,0,255))
        print(farbe.red(), farbe.green(), farbe.blue())

    def button11_clicked(self):
        font = QFontDialog.getFont()

    def button12_clicked(self):
        d = Dialog(self)
        d.exec()

app = QApplication([])
f = Fenster()
app.exec()
```

Kapitel 8 – GUI Programmierung: QtDesigner

Vorlesung

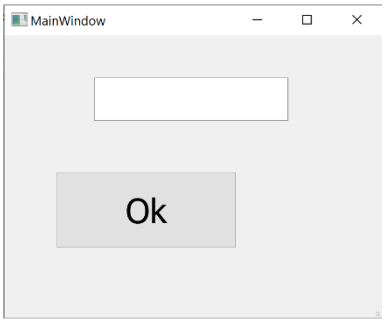
- **mygui.py**

```
from PyQt5.QtWidgets import *
from PyQt5.uic import *


def hello():
    print("Button wurde geklicked!")
    fenster.lineEdit.setText("Ok!!!")

app = QApplication([])

fenster = loadUi("Kapitel_08/gui.ui")
fenster.show()


fenster.meinSuperButton.clicked.connect(hello)

app.exec()
```



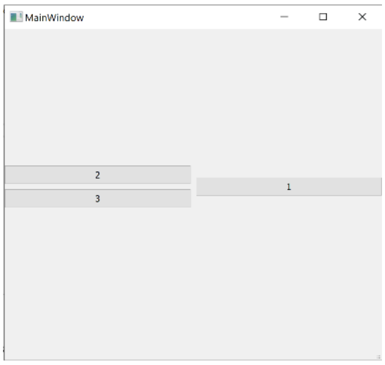
- **mygui2.py**

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

class MeinFenster(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("Vorlesung_Files von Christen/Kapitel_08/gui.ui", self)

        self.meinSuperButton.clicked.connect(self.button_click)

        self.show()

    def button_click(self):
        print("Hello")

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

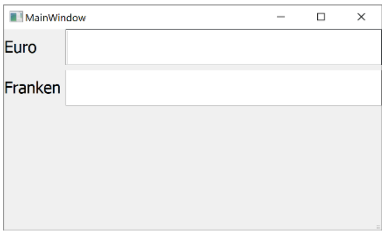


- **mygui3.py**

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

class MeinFenster(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("Vorlesung_Files von Christen/Kapitel_08/gui2.ui", self)

        self.button1.clicked.connect(self.button1_click)

        self.show()

    def button1_click(self):
        print("Hello")

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



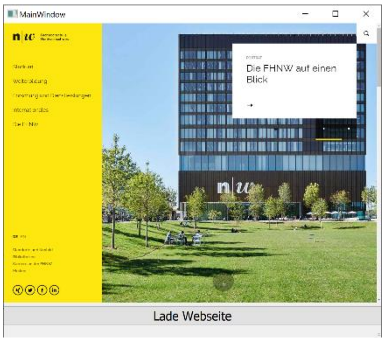
- **umrechner.py**



```python
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.uic import *

class Umrechner(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("Vorlesung_Files von Christen/Kapitel_08/gui3.ui", self)

        self.createConnects()
        self.show()

    # ------------------------------------------------------------------------

    def createConnects(self):
        #self.umrechnen.clicked.connect(self.buttonUmrechnen)
        self.euroLineEdit.textChanged.connect(self.euroEdit)

    # ------------------------------------------------------------------------

    def euroEdit(self, text):
        self.buttonUmrechnen()

    def buttonUmrechnen(self):
        euro = self.euroLineEdit.text()
        try:
            wert = float(euro)
            wert_chf = wert * 1.06
            self.frankenLineEdit.setText(str(wert_chf))
        except:
            self.frankenLineEdit.setText("ungültiger Wert")

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

- **webbrowser.py**

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

class Browser(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("Vorlesung_Files von Christen/Kapitel_08/webbrowser.ui", self)

        # Cookies erlauben
        defaultProfile = QWebEngineProfile.defaultProfile()
        defaultProfile.setPersistentCookiesPolicy(QWebEngineProfile.ForcePersistentCookies)

        self.show()

        self.pushButton.clicked.connect(self.loadPage)

    def loadPage(self):
        htmlcode = """
        <h1>Hello World</h1>
        Dies ist eine Website<br/>
        Bla bla bla
        """
        #self.webEngineView.setHtml(htmlcode)

        self.webEngineView.load(QUrl("https://www.fhnw.ch"))


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



Kapitel 9 – Matplotlib & Qt

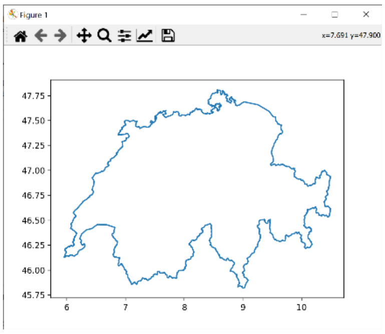
Vorlesung
KEINE

Kapitel 10 – Projektionen und Vektordaten, Teil 1: Shapely

Vorlesung

- **geometrie.py**

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

FHNW = Point([47.534862756727605, 7.641589461536574])
BSL = Point([47.598829, 7.529104])

file = open("Meine File-Lessons/Kapitel_10/schweiz.wkt")
schweiz_zeichenkette = file.read()
file.close()

schweiz = shapely.wkt.loads(schweiz_zeichenkette)

for geometry in schweiz.geoms:
    x,y = geometry.exterior.xy
    plt.plot(x,y)
plt.show()
```

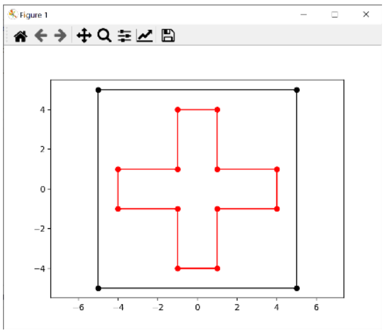


- **geometrie2.py**

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

FHNW = Point([7.641589461536574, 47.534862756727605])
BSL = Point([7.529104, 47.598829])

file = open("Meine File-Lessons/Kapitel_10/schweiz.wkt")
schweiz_zeichenkette = file.read()
file.close()

schweiz = shapely.wkt.loads(schweiz_zeichenkette)

if FHNW.within(schweiz):
    print("Die FHNW ist in der Schweiz.")      ------------- Output ------------------>      Die FHNW ist in der Schweiz.

if not BSL.within(schweiz):
    print("Starbucks am Basel Airport ist ausserhalb er Schweiz.")    -------- Output -------------> Starbucks am Basel
                                                                       Airport ist ausserhalb er Schweiz.
```

- **geometrie3.py**

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

wkt1 = "POLYGON (( -5 -5, 5 -5, 5 5, -5 5, -5 -5))"
wkt2 = "POLYGON ((1 -1, 4 -1, 4 1, 1 1, 1 4, -1 4, -1 1, -4 1, -4 -1, -1 -1, -1 -4, 1 -4, 1 -1))"

quadrat = shapely.wkt.loads(wkt1)
kreuz = shapely.wkt.loads(wkt2)

if kreuz.within(quadrat):
    print("Alles OK")

x1,y1 = quadrat.exterior.xy
x2,y2 = kreuz.exterior.xy

plt.plot(x1,y1, "ko-")
plt.plot(x2,y2, "ro-")
plt.axis("equal")

plt.show()

kreuzquadrat = kreuz.union(quadrat)
print(kreuzquadrat.wkt)

s2 = kreuz.intersection(quadrat)

s3 = kreuz.symmetric_difference(quadrat)

print(s3.wkt)

# plt.plt(x3,y3, "go-")
```



- **postleitzahlen.py**

```
from pyproj import Transformer

transformer = Transformer.from_crs('EPSG:2056', 'EPSG:4326')
t = Transformer.from_crs('EPSG:2056', 'EPSG:4326')

file = open("Meine File-Lessons/Kapitel_10/PLZO_CSV_LV95.csv", encoding="utf-8")

next(file)

# Ortsname; PLZ; Zusatzziffer; Gemeindename; BFS-Nr; Kantonskürzel; E; N; Sprache
for data in file:
    data =data.rstrip().split(";")

    name = data[0]
    E = float(data[6])
    N = float(data[7])

    resultat = t.transform(E,N)

    print(name, resultat)
```

```
Output:

Caviano (46.10716783267641, 8.76636015545509)
Piazzogna (46.1351214949252, 8.824928452154456)
Contone (46.14806740373982, 8.928681091984423)
Corippo (46.235765959714406, 8.840927326223506)
Vogorno (46.2226255892587, 8.85864040656365)
Lavertezzo (46.257208366193076, 8.835991748691933)
Brione (Verzasca) (46.29676823555998, 8.790302638961945)
Gerra (Verzasca) (46.31890735350791, 8.798422685674254)
Frasco (46.3402289193318, 8.802104599441554)
Sonogno (46.349825819278266, 8.786155280271315)
Aigle (46.317281532455375, 6.967618108130719)
…
…
…
```

- **transformation.py**

```
from pyproj import Transformer

t84 = Transformer.from_crs('EPSG:2056', 'EPSG:4326')
t95 = Transformer.from_crs('EPSG:4326', 'EPSG:2056')

"""
EPSG: 4326          Geografisches WGS84 (3D)
EPSG: 3857          Web-Mapping Mercator (z.B. Google Maps)

EPSG: 21781         CH1903/LV03
EPSG: 2056          CH1903r/LV95
"""
URL = "www.epsg.io"

resultat1 = t84.transform(2_600_000, 1_200_000)  #  _  Underscore kann bei Zahlen zur besseren lesbarkeit verwendet werden.
resultat2 = t95.transform(46.95108277187109, 7.43863242087181)

test = resultat1[0]

print(type(test))

#print(resultat1[0])
#print(resultat2)
```

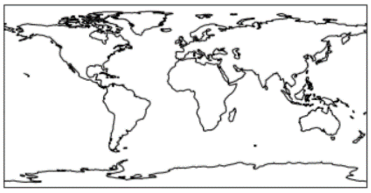
Kapitel 11 – Folium & GeoPandas

Kapitel 12 – Projektionen & Vektordaten, Teil 2: cartopy

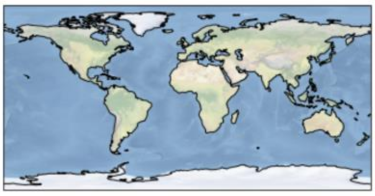
Vorlesung

- Winkeltreue (konforme) Projektionen
  - z.B: Mercator Projektion
- Flächentreue (äquivalente) Projektionen
  - z.B: Mollweide, Wagner VII
- Vermittelnde (aphylaktische) Projektionen
  - z.B: Robinson Projektion
- Längentreue (äquidistante) Projektionen
  - z.B: Mittabstandtreue Azimutalprojektion

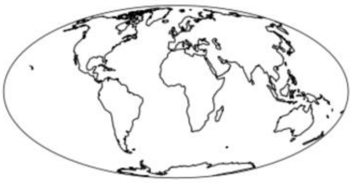
```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt

ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
plt.show()
```



```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt

ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
ax.stock_img()
plt.show()
```



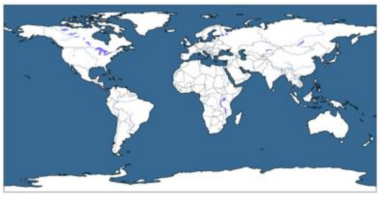
```
ax = plt.axes(projection=ccrs.Mollweide())
ax.coastlines()
plt.show()
```



Features der Standardkarte einblenden

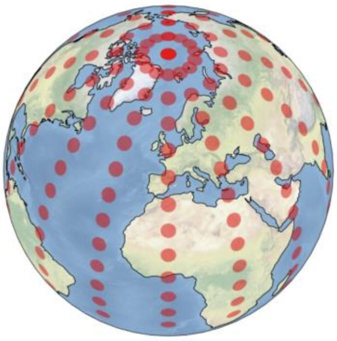
Beim ersten Aufruf werden die Daten heruntergeladen. Die "Warnung" kann ignoriert werden

```
import cartopy.feature as cfeature

plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())

ax.add_feature(cfeature.LAND, color="white")
ax.add_feature(cfeature.OCEAN, color="#006994")
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS, linestyle=':')  #
ax.add_feature(cfeature.LAKES, alpha=0.5, color="blue")
ax.add_feature(cfeature.RIVERS)
plt.show()
```

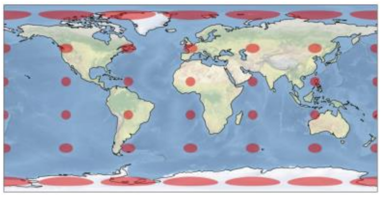


Tissot Indikatrix

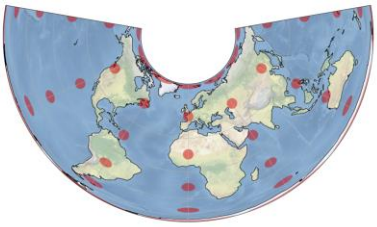
```
import cartopy.feature as cfeature
import numpy as np

plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.Orthographic(central_longitude=0.0, central_latitude=45.0))

ax.stock_img()
ax.coastlines()


lons = np.linspace(-180,180,15)
lats = np.linspace(-90,90,20)

ax.tissot(facecolor='red', alpha=0.5, rad_km=300, lons=lons, lats=lats)

plt.show()
```



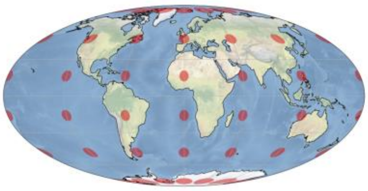
```
plt.figure(figsize=(15, 9))

ax = plt.axes(projection=ccrs.PlateCarree())

ax.set_global()
ax.stock_img()
ax.coastlines()

ax.tissot(facecolor='red', alpha=0.5)
plt.show()
```



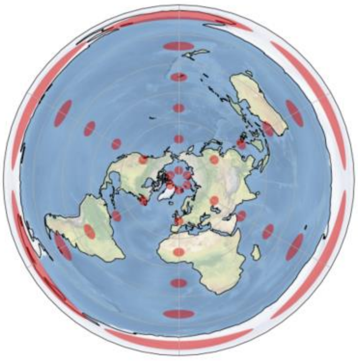
```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.AlbersEqualArea())

ax.stock_img()
ax.coastlines()
ax.gridlines()

ax.tissot(facecolor='red', alpha=0.5)
plt.show()
```



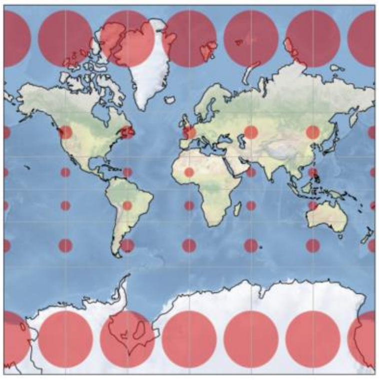
```
plt.figure(figsize=(15, 9))

ax = plt.axes(projection=ccrs.Mollweide())
ax.stock_img()
ax.coastlines()
ax.gridlines()

ax.tissot(facecolor='red', alpha=0.5)
plt.show();
```



```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.AzimuthalEquidistant(central_latitude=90))

ax.stock_img()
ax.coastlines()
ax.gridlines()
ax.tissot(facecolor='red', alpha=0.5)
plt.show();
```

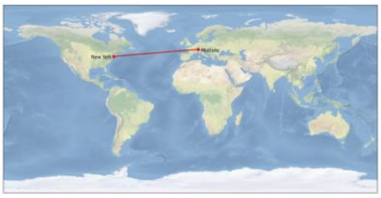


```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.Mercator(min_latitude=-85.0, max_latitude=85.0))

ax.stock_img()
ax.coastlines()
ax.gridlines()
ax.tissot(facecolor='red', alpha=0.5)
plt.show();
```



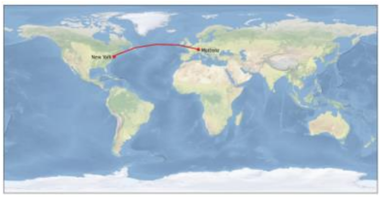
Zeichnen auf der Karte

```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()

latMuttenz, lngMuttenz = 47.534833023418976, 7.6418778566047125
latNewYork, lngNewYork = 40.68925850566851, -74.04453971488435


plt.text(lngNewYork-2, latNewYork-2, 'New York', horizontalalignment='right',
transform=ccrs.PlateCarree())
plt.text(lngMuttenz+20, latMuttenz-2, 'Muttenz', horizontalalignment='right',
transform=ccrs.PlateCarree())

plt.plot([lngMuttenz, lngNewYork], [latMuttenz, latNewYork], color='red', linewidth=2,
marker='o', transform=ccrs.PlateCarree())
plt.show()
```



Transformation: Geodätisch

```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()

latMuttenz, lngMuttenz = 47.534833023418976, 7.6418778566047125
latNewYork, lngNewYork = 40.68925850566851, -74.04453971488435


plt.text(lngNewYork-2, latNewYork-2, 'New York', horizontalalignment='right',
transform=ccrs.Geodetic())
plt.text(lngMuttenz+20, latMuttenz-2, 'Muttenz', horizontalalignment='right',
transform=ccrs.Geodetic())

plt.plot([lngMuttenz, lngNewYork], [latMuttenz, latNewYork], color='red', linewidth=2,
marker='o', transform=ccrs.Geodetic())
plt.show()
```



Geodätische Linie mit pyproj berechnen

```
import pyproj
import numpy as np

g = pyproj.Geod(ellps='WGS84')

latMuttenz, lngMuttenz = 47.534833023418976, 7.6418778566047125
latNewYork, lngNewYork = 40.68925850566851, -74.04453971488435

lnglat = g.npts(lngMuttenz,latMuttenz, lngNewYork, latNewYork, 20)

# erster und letzter Punkt hinzufügen
lnglat = np.array([(lngMuttenz, latMuttenz)] + lnglat +  [(lngNewYork, latNewYork)],
dtype=np.float64)

print(lnglat)
```

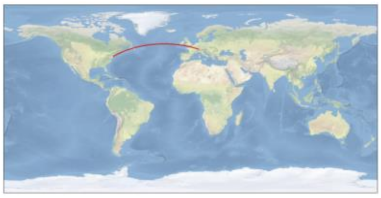
```
[[  7.64187786  47.53483302]
 [  3.9840946   48.63946998]
 [  0.1722679   49.62276528]
 [ -3.78543206  50.47548158]
 [ -7.8759829   51.18886098]
 [-12.08129479  51.75498435]
 [-16.37837788  52.16714725]
 [-20.73996119  52.42021733]
 [-25.13556591  52.51093269]
 [-29.53295367  52.43810289]
 [-33.8997926   52.20268456]
 [-38.20533514  51.80772053]
 [-42.42189904  51.25815147]
 [-46.52598714  50.56052642]
 [-50.49895725  49.72264985]
 [-54.32723661  48.75320589]
 [-58.0021438   47.66139578]
 [-61.51942119  46.45661528]
 [-64.87859239  45.14818767]
 [-68.08224822  43.74515819]
 [-71.13534094  42.25614766]
 [-74.04453971  40.68925851]]
```

```
lng = lnglat[:,0]
lat = lnglat[:,1]

print(lng)
print(lat)
```

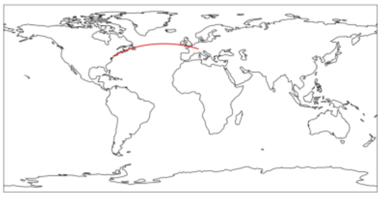
```
[  7.64187786   3.9840946    0.1722679   -3.78543206  -7.8759829
 -12.08129479 -16.37837788 -20.73996119 -25.13556591 -29.53295367
 -33.8997926  -38.20533514 -42.42189904 -46.52598714 -50.49895725
 -54.32723661 -58.0021438  -61.51942119 -64.87859239 -68.08224822
 -71.13534094 -74.04453971]
[47.53483302 48.63946998 49.62276528 50.47548158 51.18886098 51.75498435
 52.16714725 52.42021733 52.51093269 52.43810289 52.20268456 51.80772053
 51.25815147 50.56052642 49.72264985 48.75320589 47.66139578 46.45661528
 45.14818767 43.74515819 42.25614766 40.68925851]
```

Karte mit geodätischer Linie aus pyproj zeichnen

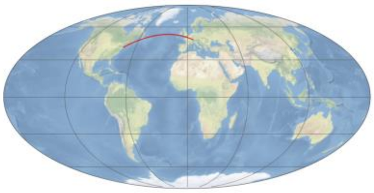
```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())

ax.stock_img()

plt.plot(lng, lat, 'r-', linewidth=2)  # 'ro-' für einzelne Punkte
plt.show()
```



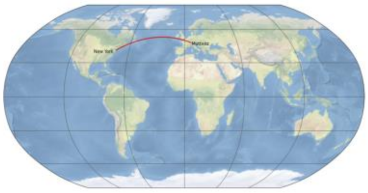
```
plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())

ax.coastlines()
ax.set_global()

plt.plot(lng, lat, 'r-', linewidth=2)  # 'ro-' für einzelne Punkte
plt.show()
```



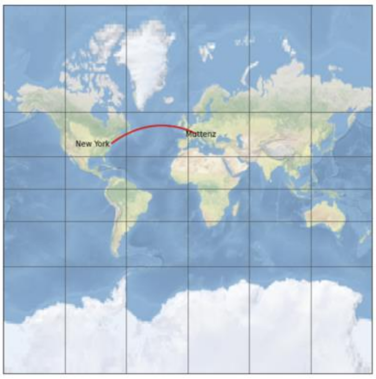
```
plt.figure(figsize=(15, 9))

ax = plt.axes(projection=ccrs.Mollweide())
ax.stock_img()
ax.gridlines(color="#555555")

plt.plot(lng, lat, 'r-', linewidth=2, transform=ccrs.Geodetic())

plt.show()
```



```
plt.figure(figsize=(15, 9))

ax = plt.axes(projection=ccrs.Robinson())
ax.stock_img()
ax.gridlines(color="#555555")


plt.text(lngNewYork-2, latNewYork-2, 'New York', horizontalalignment='right',
transform=ccrs.Geodetic())
plt.text(lngMuttenz+20, latMuttenz-2, 'Muttenz', horizontalalignment='right',
transform=ccrs.Geodetic())
plt.plot(lng, lat, 'r-', linewidth=2, transform=ccrs.Geodetic())

plt.show()
```

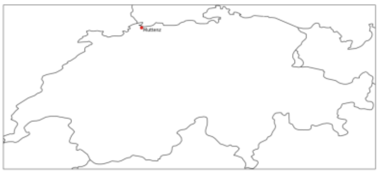


```
plt.figure(figsize=(15, 9))

ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)
ax.gridlines(color="#555555")
ax.stock_img()

plt.text(lngNewYork-2, latNewYork-2, 'New York', horizontalalignment='right',
transform=ccrs.Geodetic())
plt.text(lngMuttenz+20, latMuttenz-2, 'Muttenz', horizontalalignment='right',
transform=ccrs.Geodetic())
plt.plot(lng, lat, 'r-', linewidth=2, transform=ccrs.Geodetic())

plt.show()
```



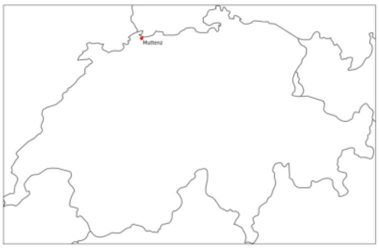
Ausschnitt (extent)

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
import cartopy.feature as cf

plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())

ax.coastlines()
ax.add_feature(cf.BORDERS)
ax.set_extent([5.96,10.49,45.82,47.81])

latMuttenz, lngMuttenz = 47.534833023418976, 7.6418778566047125

plt.plot(lngMuttenz, latMuttenz, 'ro', transform=ccrs.Geodetic())
plt.text(lngMuttenz+0.02, latMuttenz-0.05, 'Muttenz', transform=ccrs.Geodetic())

plt.show()
```



Mit Web-Merctor (EPSG:3857)

```
import cartopy.feature as cf

plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)

ax.coastlines()
ax.add_feature(cf.BORDERS)
ax.set_extent([5.96,10.49,45.82,47.81])

latMuttenz, lngMuttenz = 47.534833023418976, 7.6418778566047125

plt.plot(lngMuttenz, latMuttenz, 'ro', transform=ccrs.Geodetic())
plt.text(lngMuttenz+0.02, latMuttenz-0.05, 'Muttenz', transform=ccrs.Geodetic())

plt.show()
```



Kantone mit Cartopy

```
import geopandas as gpd

kantone = gpd.read_file("daten/gemeindegrenzen/ggg_2021-LV95/shp/g1k21.shp", encoding="utf-8")
```

Nach WGS84 transformieren und als Shapefile speichern:

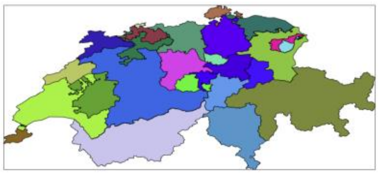
```
kantonewgs84 = kantone.to_crs("EPSG:4326")
kantonewgs84.to_file("daten/kantoneWGS84.shp", driver="Shapefile", encoding="utf-8")
```

Cartopy Shapereader

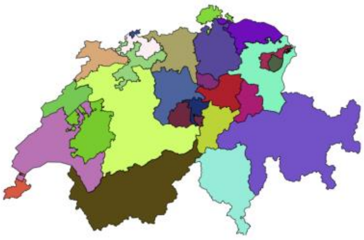
```
import cartopy.io.shapereader as shpreader

reader = shpreader.Reader("daten/kantoneWGS84.shp")

for record in reader.records():
    print(record.attributes["KTNAME"], str(record.geometry)[0:100])
```

```
Zürich POLYGON ((8.669605788370836 47.67475467340421, 8.676874952790913 47.6718171838336,
8.677959703624504 Bern / Berne MULTIPOLYGON (((7.09126653424764 46.90346257216019,
7.09741800287087 46.89996396287859, 7.1032224250 Luzern POLYGON ((8.258595258875983
47.28719707625456, 8.261630148036334 47.28254304580119, 8.27065758064512
…
```

```
from cartopy.feature import ShapelyFeature
import cartopy.io.shapereader as shpreader
import random


plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.PlateCarree())


ax.set_extent([5.96,10.49,45.82,47.81])

reader = shpreader.Reader("daten/kantoneWGS84.shp")

for record in reader.records():
    r = random.random()
    g = random.random()
    b = random.random()
    shape_feature = ShapelyFeature([record.geometry], ccrs.PlateCarree(), fc=(r,g,b),
ec='black', lw=1)
    ax.add_feature(shape_feature)


plt.show()
```



```
from cartopy.feature import ShapelyFeature
import cartopy.io.shapereader as shpreader
import random


plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)


ax.set_extent([5.96,10.49,45.82,47.81])

reader = shpreader.Reader("daten/kantoneWGS84.shp")

for record in reader.records():
    r = random.random()
    g = random.random()
    b = random.random()
    shape_feature = ShapelyFeature([record.geometry], crs=ccrs.PlateCarree(), fc=(r,g,b),
ec='black', lw=1)
    ax.add_feature(shape_feature)


plt.box(False)
plt.show()
```



Alternative: Shapefile mit GeoPandas lesen

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

Zugriff auf Daten mit iloc

```
df.iloc(0)[i]["ATTRIBUT"]
```

```
kantone.iloc(0)[0]
```

```
KTNR                                                          1
KTNAME                                                   Zürich
GRNR                                                          4
AREA_HA                                                  172894
X_MIN                                                    669245
X_MAX                                                    716900
Y_MIN                                                    223896
Y_MAX                                                    283343
X_CNTR                                                   691800
Y_CNTR                                                   252000
Z_MIN                                                       330
Z_MAX                                                      1291
Z_AVG                                                       533
Z_MED                                                       505
E_MIN                                                   2669245
E_MAX                                                   2716900
N_MIN                                                   1223896
N_MAX                                                   1283343
E_CNTR                                                  2691800
N_CNTR                                                  1252000
Shape_Leng                                        284843.337695
Shape_Area                                        1729058104.04
geometry      POLYGON ((8.669605788370836 47.67475467340421,...
Name: 0, dtype: object
```

```
kantone.iloc(0)[0]["geometry"]
```

Schwerpunkt des Polygons

```
centroid = kantone.iloc(0)[0]["geometry"].centroid
x = centroid.x
y = centroid.y
print(x,y)
```

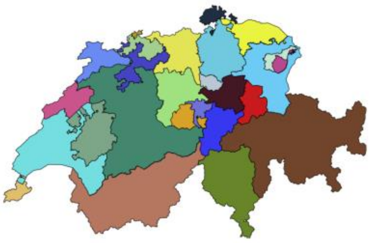
```
8.655116819326395 47.41298181137762
```

Bounding Box des Polygons

```
bounds = kantone.iloc(0)[0]["geometry"].bounds
bounds
```

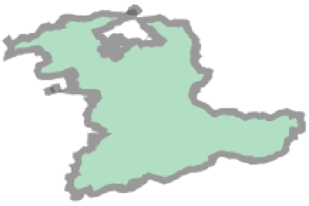
```
(8.357826373516879, 47.1594443573009, 8.985038092665208, 47.6945618446393)
```

```
from cartopy.feature import ShapelyFeature
import geopandas as gpd
import random

plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)
ax.set_extent([5.96,10.49,45.82,47.81])


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

for i in range(0, len(kantone)):
    r = random.random()
    g = random.random()
    b = random.random()
    geometry = kantone.iloc(0)[i]["geometry"]
    name = kantone.iloc(0)[i]["KTNAME"]

    shape_feature = ShapelyFeature([geometry], crs=ccrs.PlateCarree(), fc=(r,g,b), ec='black',
lw=1)
    ax.add_feature(shape_feature)


plt.box(False)
plt.show()
```



Einzelnes Polygon plotten

```
kantone = gpd.read_file("daten/kantoneWGS84.shp", encoding="utf-8")
bern = kantone.query("KTNR == 2")
bern.iloc(0)[0]["geometry"]
```



```
from cartopy.feature import ShapelyFeature
import geopandas as gpd
import random

plt.figure(figsize=(15, 9))
ax = plt.axes(projection=ccrs.GOOGLE_MERCATOR)

kantone = gpd.read_file("daten/kantoneWGS84.shp", encoding="utf-8")
bern = kantone.query("KTNR == 2")

geometry = bern.iloc(0)[0]["geometry"]
bbox = geometry.bounds

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

shape_feature = ShapelyFeature([geometry], crs=ccrs.PlateCarree(), fc="green", ec='black',
lw=1)
ax.add_feature(shape_feature)


#plt.box(False)
plt.show()
```