Modulabschlussprüfung Programmierung II

Repertorium
- div. Beispielaufgaben
- Modulabschlussprüfung

Kapitel 1 – Einführung git & GitHub
- Theorie

Kapitel 2 - Numerisches Python I
- Theorie
- Vorlesung
- Übung
- Beispielaufgaben (numpy / matplotlip)

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

Was ist eine Klassendefinition ?

Eine Klassendefinition oder Klasse beschreibt die Struktur eines Objektes, welche aus Attributen und Methoden bestehen.Es ist nur die Beschreibung eines Objekttypes und nicht das Objekt selber. Beispiel:

```
class Rechteck:
    def __init__(self, laenge, breite):
        self.laenge = laenge
        self.breite = breite

    def __str__(self):
        return f"{self.laenge}, {self.breite}"
```

Was ist eine Instanz ?

Durch die Instanz ruft man die Methoden der einzelnen Klassen auf. Es ist die Ausführung einer Klasse mit Werten für die einzelnen Attributen. Beispiel:

```
a = Rechteck(4,2)
print(a)
```

Was sind magische Methoden ?

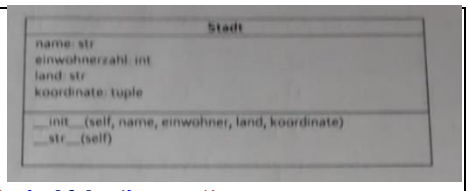
Sind spezielle Methoden, die einer Klasse die besondere Fähigkeiten geben. Sie werden automatisch (indirket) abgerufen wenn eine bestimmte Instanz aufgerufen wird.
z.B. die **__str__**. Damit wird eine Zeichenkette erstellt und wenn man dan als Instanz print(a) eingibt wird die **__str__** aufgerufen und die Werte ausgegeben welche dort definiert wurden siehe Beispiel von der Klasse. So können bestimmte ausdrücke vereinfacht ausgeführt werden.
Ein weiters Beispiel wäre **__add__** so können zwei Eingaben miteinander addiert werden. -> Eingabe z.B a + b rechnet automatisch die beiden zusammen.

Was wird automatisch aufgerufen wenn eine neue Instanz einer Klasse erstellt wird ? der Konstruktor
Wenn eine Klasse vererbt wird so muss die vererbte Klasse immer mit dem super().**__init__**() manuell aufgerufen werden damit die Attribute korrekt initialisiert werden.

```
class Rechteck:
    def __init__(self, laenge, breite):
        self.laenge = laenge
        self.breite = breite

    def __str__(self):
        return f"{self.laenge}, {self.breite}"

a = Rechteck(4,2)
print(a) # wenn print von einem Objekt muss __str__ definiert sein
```

```
---------- Output ---------->
4, 2
```

```
class Stadt:
    def __init__(self,name,einwohner,land,koordinate):
        self.name = name
        self.einwohnerzahl = einwohner
        self.land = land
        self.koordinate = koordinate

    def __str__(self):
        return f"Stadt: {self.name}, Einwohnerzahl: {self.einwohnerzahl}, Land: {self.land}, Koordinate:
{self.koordinate}"

# Die verschiededenen Variabeln könnten auch eigene Klassen definiert werden. Die seperaten Klassen könnte man
  implementieren und in den eigenständigen
# Klassen könnte man noch eigene Berechnungen oder Bedingungen definieren. Bei den Koordinaten berechnungen LV95 zu WGS84
# bei dem Land können einzelne Sachen einfacher geändert werden weil z.B. meherer Städte erfasst werden können unter dem
  gleichen Land. Das heisst
# sobald etwas beim Land geändert werden muss muss es überall gemacht werden ausser man hat eine eigene Klasse dafür
```



```python
class Student:
    def __init__(self,name,vorname,geschlecht,immu_nr, age = 0):
        self.name = name
        self.vorname = vorname
        self.geschlecht = geschlecht
        self.immu_nr = immu_nr
        self.setAge(age)
        self.mark = {}

    def setAge(self,age):
        self.age = age

    def setMark(self,topic,mark):
        self.mark[topic] = mark

    def display(self):
        print(self.name, self.vorname, self.geschlecht, self.immu_nr, self.age, self.mark)

a = Student("Walliser","Fabrice", "Male", "75666116","26")
a.setAge(27)
a.setMark("Mathe",5.9)
a.display()
------------ Output ------------------>      Walliser Fabrice Male 75666116 27 {'Mathe': 5.9}
```

setAge(...): Setzt das Alter
setMark(...): Setzt Note in einem Fach.
Es sollen beliebig viele Fächer möglich sein.
(Tipp: Dictionary)
display(...): gibt alle Informationen aus

```python
class Roman:
    def __init__(self,roman):
        self.roman = roman

    def __str__(self):
        return f"{self.roman}"

    def __add__(self, other):
        return self.int_to_Roman(self.roman_to_int(self.roman)+self.roman_to_int(other.roman))

    def __int__(self):
        return  self.roman_to_int(self.roman)
```

```python
class Punkt:
    def __init__(self,x ,y):
        self.x = x
        self.y = y
    def __str__(self):
        return f"Punkt: ({self.x}, {self.y})"

class Strecke:
    def __init__(self, A, B):
        self.a = A
        self.b = B

    def laenge(self):
        return((self.a.x-self.b.x)**2 +(self.a.y-self.b.y)**2)**0.5

    def __str__(self):
        return f"Strecke: {self.a} bis {self.b}"

class Dreieck:
    def __init__(self,A,B,C):
        self.A = A
        self.B = B
        self.C = C

    def strecken(self):
        self.a = Strecke(self.B,self.C)
        self.b = Strecke(self.A,self.C)
        self.c = Strecke(self.A,self.B)

    def flaeche(self):
        self.s = (self.a.laenge()+ self.b.laenge() + self.c.laenge()) / 2
        self.F = (self.s*(self.s-self.a.laenge())*(self.s-self.b.laenge())*(self.s-self.c.laenge()))**0.5
        return self.F

    def umfang(self):
        u = self.a.laenge() + self.b.laenge() + self.c.laenge()
        return u

    def inkreis(self):
        r = (2 * self.F) / (self.a.laenge()+self.b.laenge()+self.c.laenge())
        return r

X = Punkt(0,0)
Y = Punkt(1,0)
Z = Punkt(1,1)
d = Dreieck(X,Y,Z)

flaechedr = d.flaeche()
print(flaechedr)

udrei = d.umfang()
print(udrei)

dinkr = d.inkreis()
print(dinkr)
```

Gegeben sind auch die Klasse Punkt und Strecke *(nicht abschreiben!!)*

```python
class Punkt:
    def __init__(self, x, y):
        self.x = x
        self.y = y

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


class Strecke:
    def __init__(self, A, B):
        self.a = A
        self.b = B

    def laenge(self):
        return ((self.a.x-self.b.x)**2+(self.a.y-self.b.y)**2)**0.

    def __str__(self):
        return f"STRECKE: {self.a} bis {self.b}"
```

Implementieren Sie neben dem Konstruktor die folgenden Methoden:

`flaeche()`: Gibt die Fläche des Dreiecks zurück
`umfang()`: Gibt den Umfang des Dreiecks zurück
`inkreisradius()`: Gibt den Radius des Inkreises zurück

Hinweis:

Die Fläche F des Dreicks kann z.B. über den Satz des Heron berechnet werden:

$$s = \frac{a+b+c}{2}$$

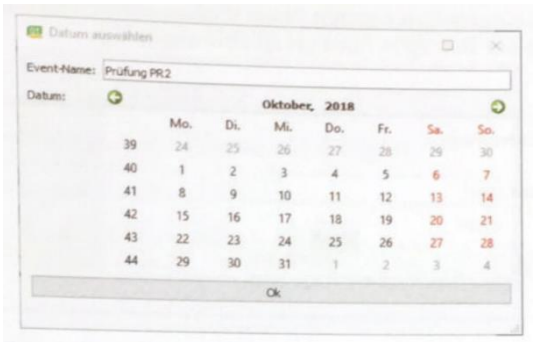
$$F = \sqrt{s(s-a)(s-b)(s-c)}$$

Wobei a,b und c die Seitenlängen des Dreiecks sind.

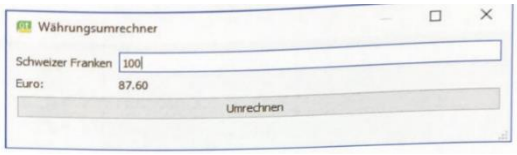
Der Inkreisradius r des Dreiecks berechnet sich mit $r = \frac{2F}{a+b+c}$

```python
class Punkt:
    def __init__(self,x ,y):
        self.x = x
        self.y = y
    def __str__(self):
        return f"Punkt: ({self.x}, {self.y})"

class Quadrat:

    def __init__(self,A,B):
        self.A = A
        self.B = B

    def umfang(self):
        self.u = (abs((self.A.x - self.B.x)))*4
        return self.u

    def flaeche(self):
        self.f = (abs((self.A.x - self.B.x)))**2
        return self.f

    def mittelpunkt(self):
        self.mx = (abs((self.A.x - self.B.x))) / 2
        self.my = (abs((self.A.y - self.B.y))) / 2
        return f"Mittelpunkt: X: {self.mx}, Y: {self.my}"

X = Punkt(0,0)
Y = Punkt(1,1)

q = Quadrat(X,Y)
um = q.umfang()
fl = q.flaeche()
mi = q.mittelpunkt()
print(um)      ------------- Output ------------------>  4
print(fl)      ------------- Output ------------------>  1
print(mi)      ------------- Output ------------------>  Mittelpunkt: X: 0.5, Y: 0.5
```

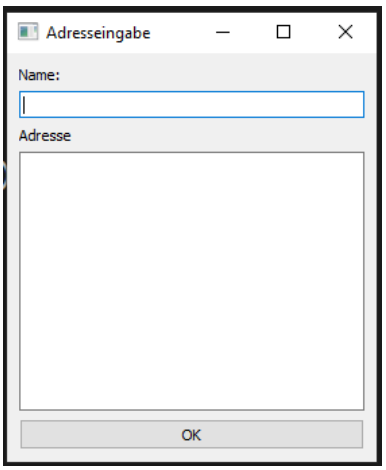
```python
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Datum auswählen")

        layout = QFormLayout()

        self.event = QLineEdit()
        self.calendar = QCalendarWidget()
        self.button = QPushButton("Ok")

        layout.addRow("Event-Name", self.event)
        layout.addRow("Datum", self.calendar)
        layout.addRow(self.button)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

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



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

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

        self.setWindowTitle("Währungsumrechner")

        layout = QFormLayout()

        self.franken = QLineEdit()
        self.euro = QLabel()
        self.umkurs = QLineEdit()
        self.umrechnen = QPushButton("Umrechnen")

        layout.addRow("Schweizer Franken", self.franken)
        layout.addRow("Umrechnungskurs", self.umkurs)
        layout.addRow("Euro", self.euro)
        layout.addRow(self.umrechnen)

        self.umrechnen.clicked.connect(self.rechner)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

    def rechner(self):
        try:
            chf = float(self.franken.text())
            kurs = float(self.umkurs.text())
            euro = chf * kurs
            self.euro.setText(str(round(euro,2)))
        except:
            QMessageBox.warning(self,"Achtung auf Eingabe", "Es müssen Zahlen eingegeben werden")

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



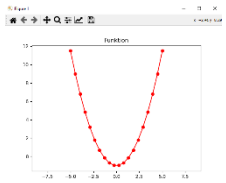

```python
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

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

        self.setWindowTitle("Adresseingabe")

        layout = QFormLayout()

        self.name = QLabel()
        self.adresse = QLabel()
        self.name_eingabe = QLineEdit()
        self.adresse_eingabe = QTextEdit()
        self.button = QPushButton("OK")

        layout.addRow("Name:", self.name)
        layout.addRow(self.name_eingabe)
        layout.addRow("Adresse", self.adresse)
        layout.addRow(self.adresse_eingabe)
        layout.addRow(self.button)

        center = QWidget()
        center.setLayout(layout)

        self.setCentralWidget(center)
        self.show()

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



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

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

        loadUi("showmap.ui", self)

        self.convertButton.clicked.connect(self.convert)

        self.show()

    def convert(self):
        try:
            self.chf = float(lineEditCHF)
            self.euro = 0.999 * self.ch
            lineEdit.setText(str(self.euro))

        except:
            QMessageBox.warning(self,"Achtung", "Esm müssen Zahlen eingegeben werden")

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

Datum
jahr: int monat: int tag: int
__init__(self, Jahr, Monat, Tag) __str__(self) heute() : Datum

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

x = np.linspace(-5,5,20)
y = (0.5 * x**2) -1
plt.plot(x,y,"ro-")
plt.title("Funktion")
plt.axis("equal")
plt.show()
```



```python
class Gebaeude:
    def __init__(self, strasse, hausnr):
        self.strasse = strasse
        self.nr = hausnr

class Bauernhaus(Gebaeude):
    def __init__(self, strasse, hausnr, tiere, traktor):
        super().__init__(strasse, hausnr)
        self.tiere = tiere
        self.traktor = traktor

    def __str__(self):
        return f"Adresse: {self.strasse} {self.nr}, Tiere: {self.tiere}, Traktor: {self.traktor}"

class Wohngebaeude(Gebaeude):
    def __init__(self, strasse, hausnr, zimmer, moebel):
        super().__init__(strasse, hausnr)
        self.zimmer = zimmer
        self.moebel = moebel



    def __str__(self):
        return f"Adresse: {self.strasse} {self.nr}, Anzahl Zimmer: {self.zimmer}, Möbiliert: {self.moebel}"


bauernhof = Bauernhaus("Feldweg", "123b", "Schafe, Ziegen", "John Deere")
print(bauernhof)

wohnung = Wohngebaeude("Dorfstrasse", "37", 5, True)
print(wohnung)
```

```
Adresse: Feldweg 123b, Tiere: Schafe, Ziegen, Traktor: John Deere
Adresse: Dorfstrasse 37, Anzahl Zimmer: 5, Möbiliert: True
```

```python
import datetime

class Datum:
    def __init__(self, Jahr = 2021, Monat = 1, Tag =1):
        self.Jahr = Jahr
        self.Monat = Monat
        self.Tag = Tag

    def __str__ (self):
        return f"Tag: {self.Tag}, Monat: {self.Monat}, Jahr: {self.Jahr}"

    def heute(self):
        t = datetime.datetime.now()
        self.Jahr = t.year
        self.Monat = t.month
        self.Tag = t.day
```

```python
class Meter:
    def __init__ (self,meter):
        self.meter = meter/1000

    def __str__ (self):
        return str(self.meter) + "m"

class Dezimeter:
    def __init__ (self, dezimeter):
        self.wert = dezimeter /100
        self.dezimeter = dezimeter

    def __str__ (self):
        return str(self.dezimeter) +"dm"

    def dezimal():
        return self.wert

    def __add__ (self,other):
        return self.dezimal() + other()

    def __radd__ (self, other):
        return self.dezimal()+other()
```

In Python sollen die Längenmasse als Klassen implementiert werden, also z.B. "Millimeter", "Zentimeter", "Dezimeter", "Meter" und "Kilometer". Dabei soll es möglich sein, Instanzen dieser Klassen miteinander zu addieren, subtrahieren, multiplizieren und dividieren – dabei soll als Resultat wiederum ein geeignetes Längenmass erzeugt werden, also beispielsweise:

```
a = Meter(5.1)     # 5.1 Meter
b = Dezimeter(10) # 10 Dezimeter

r = a + b
```

Beim Aufruf von print, soll das Längenmass angegeben werden, also z.B.

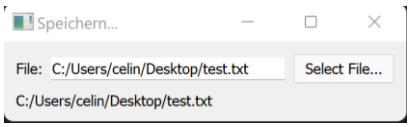
```
print(a)     # Ausgabe:   5.1 m
print(b)     # Ausgabe:   10 dm
```

Implementieren Sie dies vorerst für die Klassen „Meter" und „Dezimeter".

```python
import math
import re
from turtle import rt

class Punkt:
    def __init__(self,x ,y):
        self.x = x
        self.y = y

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

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

class Kreis(Punkt):
    def __init__ (self, M=Punkt(0,0), r=1):
        super().__init__(M, r)

        self.Mittelpunkt = M
        self.radius = r

    def mittelpunkt(self):
        return self.Mittelpunkt

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

    def flaeche(self):
        return self.radius**2*math.pi

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

k = Kreis(Punkt(1,1), 4)

print(k)                  ------------- Output ------------------>    Kreis: Punkt: (1, 1), Radius: 4
print(k.mittelpunkt())    ------------- Output ------------------>    (1, 1)
print(k.umfang())         ------------- Output ------------------>    25.132741228718345
print(k.flaeche())        ------------- Output ------------------>    50.26548245743669
```

```
import math

class Kreis():
def __init__(self, x, y, r):
self.x = x
self.y = y
self.r = r

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

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

def mittelpunkt(self):
return str(self.x), str(self.y)

k = Kreis(10,-1.5)

print(k.umfang())
print(k.flaeche())
print(k.mittelpunkt())
```

```python
from fileinput import filename
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *

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

        self.createLayout()
        self.createConnects()

    def createLayout(self):
        self.setWindowTitle("Speichern...")

        layout = QHBoxLayout()
        layoutV = QVBoxLayout()

        # Widgets erstellen
        self.button = QPushButton("Select File...")
        self.pfad = QLineEdit()
        self.file = QLabel("File:")
        self.label = QLabel("")

        # Widgets dem Layout hinzufügen
        layout.addWidget(self.file)
        layout.addWidget(self.pfad)
        layout.addWidget(self.button)

        layoutV.addLayout(layout)
        layoutV.addWidget(self.label)

        # CentralWidget
        center = QWidget()
        center.setLayout(layoutV)

        self.setCentralWidget(center)
        self.show()

    # Aktionen definieren
    def createConnects(self):
        self.button.clicked.connect(self.buttonClicked)

    def buttonClicked(self):
        filename, filter = QFileDialog.getSaveFileName(self, "Datei speichern", "C:/data/", "Textdatei [.txt]")
        if filename != "":
            self.pfad.setText(filename)
            self.label.setText(filename)

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




```python
# Aufgabe 5
# Gegeben ist eine Liste data bestehens aus Tupeln mit Kantonsnamen und der Wahlbeteiligung
# in Prozent der Nationalratswahlen. Daraus wird mit Pandas ein Dataframe erstellt, und die
# die Spalten bekommen die Namen "kanton" und "wahlbeteiligung"

# (Screenshot des DataFrames)

# Wie kann ein neues DataFrame erstellt werden für alle Kantone mit Wahlbeteiligung grösser als 50%?

df [df["wahlbeteiligung"]>50]
```

```python
# Aufgabe 6
# Schreiben Sie ein Python-Programm, welches überprüft, ob das Modul "numpy" installiert ist.

try:
    import numpy
except ImportError:
    print("numpy ist nicht installiert")
```

```
# Aufgabe 7
# a) Was geschieht, wenn die ungerstehnede Zelli im Juypter Notebook ausgeführt wird?

from matplotlib.pyplot import *$
axis("equal")
plot ([1,2,3,4,5], [3,3,3,2,2], "bo-")
show

Es wird ein Plot-Fenser erstellt, in dem die Punkte als blaue Kreise
erscheinen und verbunden sind mit einer durchgezoenern Linie.
[1,2,3,4,5] x-Werte
[3,3,3,2,2] y-Werte
mit axis("equal") sind die Koordinatenachsen gleich skaliert

# b) Und was geschieht, wenn die folgenden Zelle ausgeführt wird?
import numpy as np
a = np.array([1,2,3], dtype=np.float)
b = np.array([2,3,4], dtype=np.float)
a*b

Die beiden Arrays werden elementweise multipoizeirt.
Das Resultat ist eine Array [2,6,12]
Der Datentyp des Elements ist float.
```