**Modulabschlussprüfung Programmierung II**

[**Repertorium**](#_Repertorium)

* div. Beispielaufgaben
* Modulabschlussprüfung

[**Kapitel 1 – Einführung git & GitHub**](#_Kapitel_1_–)

* Theorie

[**Kapitel 2 - Numerisches Python I**](#_Kapitel_2_-)

* Theorie
* Vorlesung
* Übung
* Beispielaufgaben (numpy / matplotlip)

[**Kapitel 3 – Objektorientierung, Teil 1**](#_Kapitel_3_–)

* Theorie
* Vorlesung
* Übung

[**Kapitel 4 – Objektorientierung, Teil 2**](#_Kapitel_4_–)

* Theorie
* Vorlesung
* Übung

**[Kapitel 5 – Objektorientierung, Teil 3](#_Kapitel_5_–)**

**[Theorie](#_Kapitel_5_–)**

* Vorlesung
* Übung
* Beispielaufgaben (class)

[**Kapitel 6 – GUI Programmierung, Teil 1**](#_Kapitel_6_–)

* Theorie
* Vorlesung
* Übung
* Zusammenfassung GUI

[**Kapitel 7 – GUI Programmierung, Teil 2**](#_Kapitel_7_–)

* Theorie
* Vorlesung
* Übung

[**Kapitel 8 – GUI Programmierung:  
QtDesigner**](#_Kapitel_8_–)

* Theorie
* Vorlesung
* Übung
* Beispielaufgaben (qt)

[**Kapitel 9 – Matplotlib & Qt**](#_Kapitel_9_–)

* Theorie
* Vorlesung
* Übung

[**Kapitel 10 – Projektionen und Vektordaten, Teil 1: Shapely**](#_Kapitel_10_–)

* Theorie
* Vorlesung
* Übung

[**Kapitel 11 – Folium & GeoPandas**](#_Kapitel_11_–)

* Übung
* Beispielaufgaben (panda)

[**Kapitel 12 – Projektionen & Vektordaten, Teil 2: cartopy**](#_Kapitel_12_–)

* 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

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

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

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

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung        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}

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)

Ein Bild, das Text, Quittung, Dokument enthält.

Automatisch generierte Beschreibungclass 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)

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

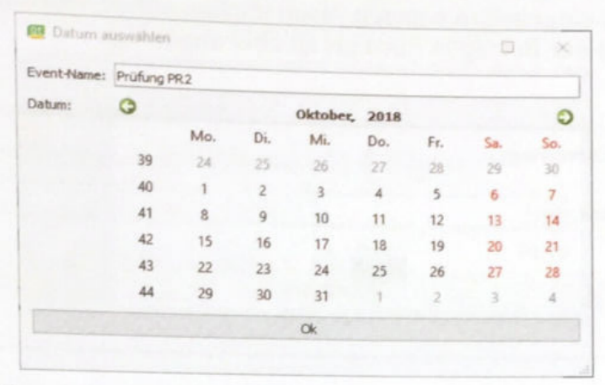
print(fl) ------------- Output ------------------> 1

print(mi) ------------- Output ------------------> Mittelpunkt: X: 0.5, Y: 0.5

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()

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import \*

Ein Bild, das Text enthält.

Automatisch generierte Beschreibungclass 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()

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import \*

Ein Bild, das Text enthält.

Automatisch generierte Beschreibungclass 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()

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import \*

from PyQt5.uic import \*

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

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))

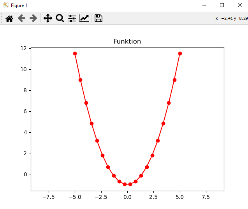
        except:

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

app = QApplication([])

fenster = ShowMap()

app.exec()

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()

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)

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

class Meter:

    def \_\_init\_\_ (self,meter):

        self.meter = meter/1000

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

    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()

import math

Ein Bild, das Text enthält.

Automatisch generierte Beschreibungimport 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

from fileinput import filename

from PyQt5.QtWidgets import \*

Ein Bild, das Text enthält.

Automatisch generierte Beschreibungfrom 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()

# 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]

# 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.