Programmieren Teil 2

Wintersemester 2024/2025

Christian Rybovic
L-TIP-24-Do-a

«Python ist eine universelle, üblicherweise interpretierte, höhere Programmiersprache. Sie hat den Anspruch, einen gut lesbaren, knappen Programmierstil zu fördern.»

Inhalt

Kompetenznachweis 1 vom 02.11.24

Für diesen Kompetenznachweis wurde eine neue Klasse namens **MultiFilter** erstellt und zur einfacheren Handhabung bereits verschiedene Lambdas vordefiniert. Die Klasse wird mit einer Liste von Zahlen aufgerufen, auf welche man zu einem späteren Zeitpunkt eine variable Anzahl von Filtern anwenden kann.

Angenommen eine Liste soll nach folgenden Bedingungen gefiltert werden:

- Nur gerade Zahlen
- Nur Zahlen grösser 20 und kleiner 60
- Nur Zahlen eines vielfachen von 3

Mit Lambda kann man nun auf mehrere Zeilen verteilt oder alles in einer Zeile die verschiedenen Filter auf die Liste anwenden. Das Ergebnis ist jedoch mit beiden Varianten eher unübersichtlich:

```
# ----- Ohne MultiFilter (1) ----- #
filtered = filter(lambda x : x % 2 == 0, numbers)
filtered = filter(lambda x : 20 < x filtered)
filtered = filter(lambda x : x % 3 == 0, filtered)
filtered = filter(lambda x : x < 60, filtered)

# ----- Ohne MultiFilter (2) ----- #
filtered = filter(lambda x : (x < 60) and (x %2 == 0) and (x > 20) and (x % 3 == 0), numbers)
```

Mithilfe der neu erstellten Klasse reduziert sich der Aufruf auf wenige Zeile, kann sehr einfach nach belieben erweitert werden und schaut übersichtlich aus. Beim Erstellen wird die Liste angegeben und beim Aufruf der Funktion «**Run**» eine oder mehrere vordefinierte (im Script ersichtlich) oder eigene Lambdas:

```
mf = MultiFilter(numbers)
filtered = mf.Run(ABOVE_20,BELOW_60,POWEROF_3,EVEN_ONLY)

print("Variante MutliFilter:", filtered)
```

Der Code für die Klasse **MultiFilter** schaut folgendermassen aus:

```
class MultiFilter:
    def __init__(self, items):
        self.items = list(items)

    def Run(self, *func):
        for f in func:
            self.items = list(filter(f, self.items))
        return self.items
```

Im Anschluss können bei Bedarf noch weitere Funktionen auf die gefilterte Liste angewendet werden (z.B. **map** oder **reduce**):

```
ADD100 = lambda x : x + 100
print("Add 100 to all items:", list(map(ADD100, filtered)))

MULTIPLY = lambda x, y : x * y
print("Multiply:", functools.reduce(MULTIPLY, filtered))
```

Kompetenznachweis 2 vom 09.11.24

Eine Klasse **User** wurde erstellt, welche Objekte mit einem Benutzernamen, Passwort und einer Mailadresse instanziiert. Die Klasse hat folgende Eigenschaften:

- Der Benutzername kann nur gelesen und nicht mehr geändert werden
- Das Passwort wird nur intern verwendet (daher ausserhalb der Klasse nicht zugreifbar)
- Die Mailadresse kann gelesen und geändert werden, für die Anpassung benötigt es aber das korrekte Passwort
- Die Klasse stellt eine Helper-Funktion für die Mailvalidierung zur Verfügung
- Keine Variable darf direkt geschrieben oder gelesen werden (Kontrolle/Sicherheit)

```python
class User:
    def __init__(self, username, password, mail):
        self.__username = username
        self.__mail = mail
        self.__password = password

    def GetMail(self):
            return self.__mail

    def SetMail(self, password, mail):
        self.__IsValidPassword(password)
        self.__mail = mail

    def __IsValidPassword(self, password):
        if(self.__password != password):
            raise Exception("Wrong password")

    def IsValidMail(mail):
        if(("@" not in mail) or ("." not in mail) or (' ' in mail)):
            return False

        if(not (0 < len(mail.split("@")[0]) <= 64) or not (0 < len(mail.split("@")[1]) <= 255)):
            return False

        return True
```

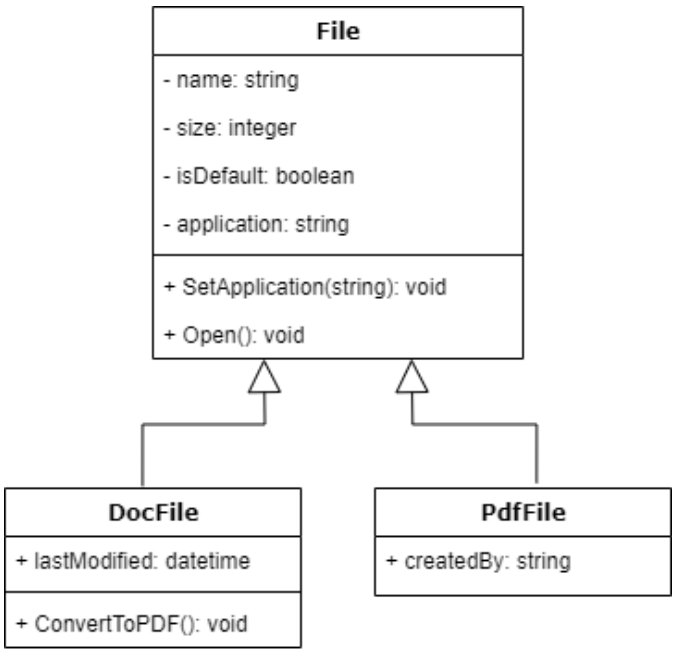
Um sicherzustellen, dass die Funktion «IsValidMail» auch in Zukunft korrekt funktioniert, wurden ein paar **UnitTests** mit dem Test-Runner «unittest» erstellt. Damit keine Flüchtigkeitsfehler schon beim Erstellen der Tests entstehen, wird mit einer Variable gearbeitet, wobei diese den Test zuerst positiv durchlaufen muss.

```python
class TestMethods(unittest.TestCase):
    #check if mail validation works as expected
    def test_mailvalidation(self):
        mail = "mail@example.ch"

        # check testmail is considered valid
        self.assertEqual(Model.User.IsValidMail(mail), True)

        # check missing "@" is detected
        self.assertEqual(Model.User.IsValidMail(mail.replace("@", "")), False)

        # check missing "." is detected
        self.assertEqual(Model.User.IsValidMail(mail.replace(".", "")), False)

        # check empty space is considered as false
        self.assertEqual(Model.User.IsValidMail(mail.replace("i", " ")), False)
```

Kompetenznachweis 3 vom 16.11.24

Zum Üben von Vererbungen in Python, wurde die Basis-Klasse **File** mit ein paar Eigenschaften und Methoden erstellt. Diese wird von den Klassen **DocFile** und **PdfFile** vererbt. Beide Klassen erweitern die Basis-Klasse um eine Eigenschaft und die DocFile-Klasse zusätzlich um eine weitere Methode. Für eine einfachere Erklärung wurde ein UML-Diagramm zur visuellen Unterstützung erstellt.

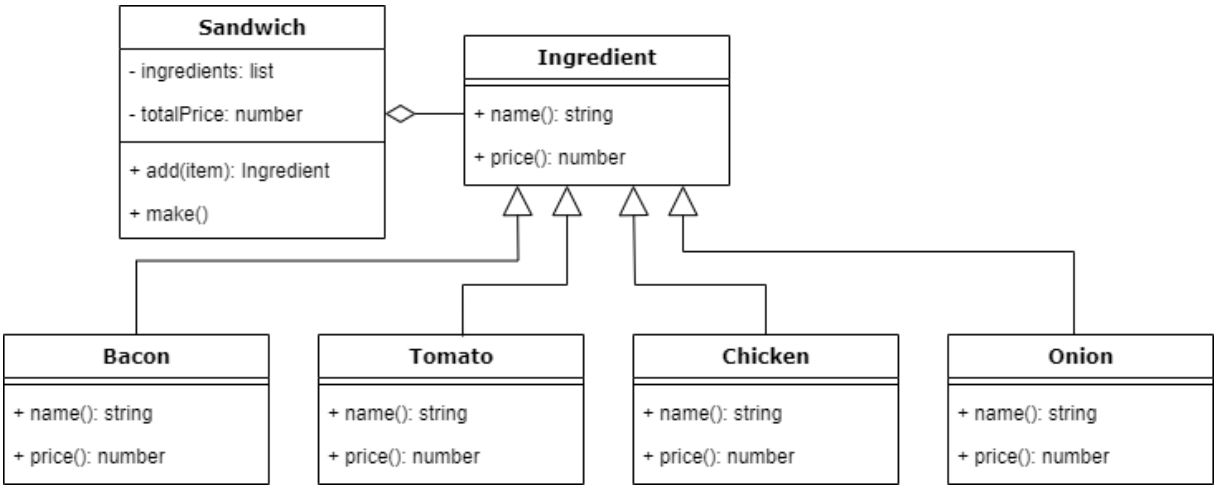
```python
class File:
    def __init__(self, name, size):
        self.__name = name
        self.__size = size
        self.__isDefault = True
        self.__application = ""

    def SetApplication(self, application):
        self.__application = application
        self.__isDefault = False

    def Open(self):
        if(self.__isDefault):
            print("Open with explorer")
        else:
            print("Open with", self.__application)

class DocFile(File):
    def __init__(self, name, size):
        self.lastModified = datetime.now()
        super().__init__(name, size)
        super().SetApplication("Office Word")

    def ConvertToPDF(self):
        print("Converting DOC to PDF")

class PdfFile(File):
    def __init__(self, name, size):
        self.createdBy = "Administrator"
        super().__init__(name, size)
```



Kompetenznachweis 4 vom 23.11.24

Das Ziel mit diesem Script ist es, ein Sandwich dynmisch mit Zutaten zu erstellen. Zuerst wurde die Basisklasse **Sandwich** erstellt. Diese hat einen Grundpreis und es können beliebig viele Zutaten hinzugefügt werden. Die Zutaten erben von der Klasse **Ingredient**, welche über abstrakte Methoden die Implementierung der Methoden **name()** und **price()** voraussetzt. Bei jeder hinzugefügten Zutat werden die Kosten zum Preis dazu gerechnet. Beim Aufruf von **make()** wird das Sandwich erstellt und der Gesamtpreis ausgegeben.

```python
class Sandwich():
    def __init__(self):
        self.__ingredients = []
        self.__totalPrice = 2.00

    def add(self, item):
        self.__ingredients.append(item)
        self.__totalPrice += item.price()
        return self

    def make(self):
        print("Making sandwich with:")
        for i in self.__ingredients:
            print( "   {0:10}+ {1}".format(i.name(), i.price()))
        print("Total: {0}\n\n".format(round(self.__totalPrice,2)))
```



Die Methode **add()** innerhalb der Sandwich-Klasse gibt mit return das Objekt selber zurück. Dadurch kann die Funktion über sogenanntes «method chaining» direkt hintereinander aufgerufen werden. So kann praktisch mit einer Zeile ein Objekt komplett erstellt werden.

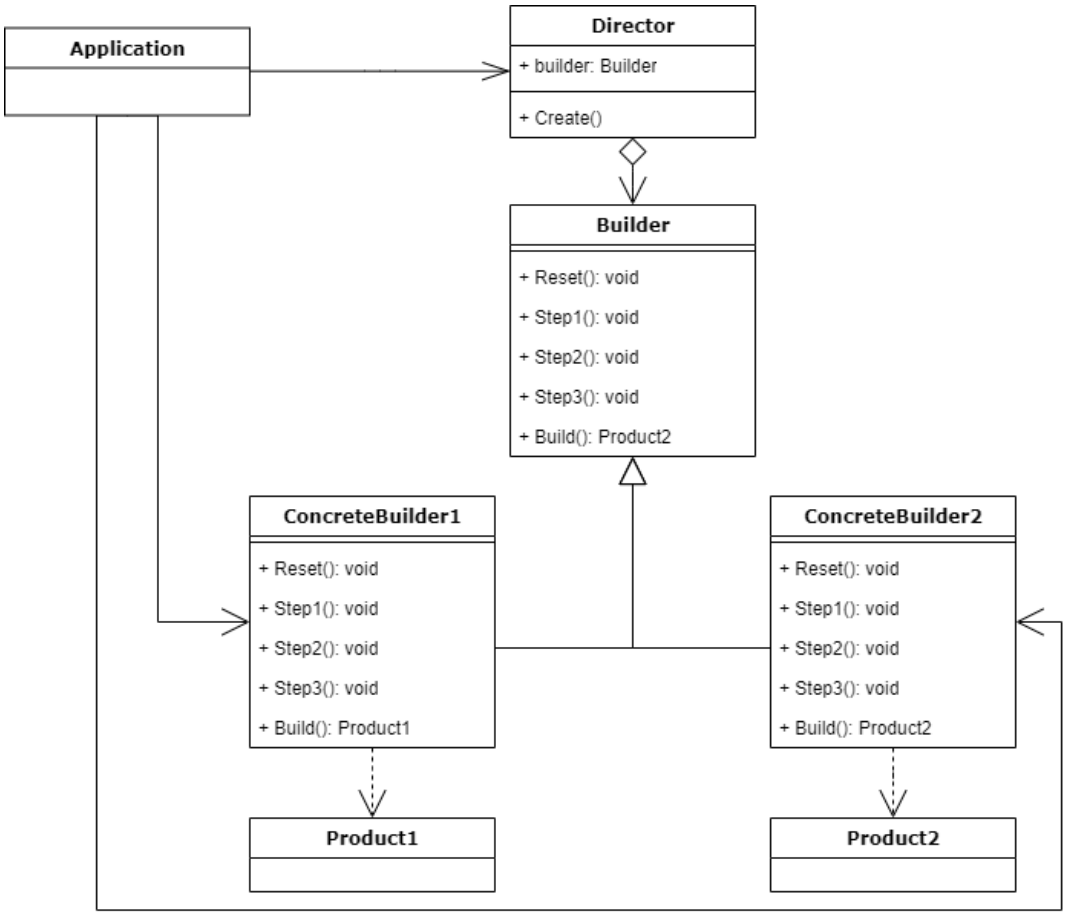
```python
# Classical
sandwich = Sandwich()
sandwich.add(Onion())
sandwich.add(Chicken())
sandwich.add(Onion())
sandwich.make()

# Method chaining
Sandwich().add(Tomato()).add(Bacon()).add(Onion()).make()
```

Kompetenznachweis 5 vom 30.11.24

Mit dem **Singleton Pattern** wird sichergestellt, dass es von einem Objekt nur eine einzige Instanz gibt. Zur Anwendung kann es bei Datenbankverbindungen, beim Schreiben in eine Datei (Logging) oder bei globalem Zugriff kommen. Das Pattern wird gerne und oft genutzt, muss aber trotzdem mit Bedacht eingesetzt werden. Eine Singleton-Klasse kann im Minimum beispielsweise folgendermassen ausschauen:

```python
class DatabaseConnection(object):
    # at object initialization
    def __new__(cls):
        # check if instance already exist, if not create new
        if not hasattr(cls, 'instance'):
            cls.instance = super(DatabaseConnection, cls).__new__(cls)
        return cls.instance

# use whenever needed (only one instance exists at anytime)
connection = DatabaseConnection()
```

Mithilfe des **Builder Design Patterns** können komplexe Objekte einfach und konform erstellt werden. Hierbei weiss der Builder, welche Möglichkeiten es gibt und der Director, wie diese dann Zusammengesetzt werden müssen. Als UML-Diagramm könnte das dann beispielsweise folgendermassen ausschauen:



Kompetenznachweis 6 vom 07.12.24

Mithilfe des **Proxy Patterns** kann sichergestellt werden, dass ein Objekt verfügbar ist bzw. die Klasse kann den Zustand und die Informationen vor der Verarbeitung prüfen (wie ein Proxy). Als Beispiel im Code wurde ein Hotel mit dem Namen «Krone» abgebildet. Ein Gast möchte ein Zimmer buchen und der Empfang (Proxyfunktion) prüft, ob und welches Zimmer frei ist. Dieses kann dem Gast dann gegeben werden und der Empfang verbucht den neuen Status.

```python
class Hotel:
    def __init__(self, name):
        self.name = name
        self.rooms = []

    def book(self, guest):
        self.rooms.append(guest)
        print("{0}: Reservation for {1}".format(self.name, guest))

class HotelProxy:
    def __init__(self, hotel):
        self.hotel = hotel

    def book(self, guest):
        if (len(self.hotel.rooms) < 3):
            self.hotel.book(guest)
        else:
            print("No free room at {0}!".format(self.hotel.name))
```

Der Proxy stellt in diesem Fall sicher, dass im Hotel nicht mehr als 3 Gäste ein Zimmer reservieren können. Ohne den Proxy würde das Hotel die Kapazität nicht prüfen und jeden Gast aufnehmen, was zur Überfüllung führen würde. Nachfolgenden werden noch die Aufrufe gezeigt und wie das Programm reagiert bzw. was vom Programm ausgegeben wird.

```python
hotel = Hotel("Hotel Krone")
proxy = HotelProxy(hotel)

proxy.book("Max")
# Hotel Krone: Reservation for Max

proxy.book("Florian")
# Hotel Krone: Reservation for Florian

proxy.book("Andrea")
# Hotel Krone: Reservation for Andrea

proxy.book("Thomas")
# No free room at Hotel Krone!
```

Kompetenznachweis 7 vom 14.12.24

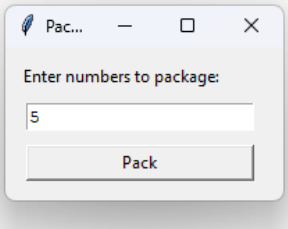
Für diesen Kompetenznachweis wurde eine **Quiz-Applikation** geschrieben. Die Fragen können über 3 mögliche Schaltflächen beantwortet werden. Bei falscher Beantwortung wird ein Dialog angezeigt, bei korrekter Beantwortung kommt die nächste Frage. Das Hintergrundbild wurde mit **Base64** kodiert, sodass dieses direkt im Programmcode eingebettet werden kann und man ausser dem Script keinen weiteren Abhängigkeiten hat.


```python
class QuestionWindow:
    def __init__(self, value):
        self.next = False
        self.number = value
        self.window = Tk()
        ...
        lblQuestion = Label(self.window, padx=30, pady=5, bg="#C9DDEE", fg="#4A6073",
                            text=questions[self.number][QUESTION],
                            font=('ariel',16,'bold'), wraplength=430 )
        lblQuestion.place(x=80, y=70)

        btnLeft = Button(self.window, text=questions[self.number][ANSWER1], width=15,
                         font=('ariel',12,'bold'), bg="#4A6073", fg="white",
                           pady=5, command=self.cmdLeft)
        btnLeft.place(x=50,y=350)
        ...
        self.window.mainloop()

    def cmdLeft(self):
        if(questions[self.number][ANSWER1] == questions[self.number][CORRECT]):
            self.solvedCorrect()
        else:
            self.solvedWrong()

    def solvedCorrect(self):
        self.next = True
        self.window.destroy()

    def solvedWrong(self):
        messagebox.showerror("Quiz", "Diese Antwort ist falsch!")

    def nextQuestion(self):
        return self.next
```

Kompetenznachweis 8 vom 21.12.24

In diesem Kompetenznachweis wurde **Threading** bzw. **Multi-Threading** bearbeitet. Hierfür wurde ein Programm erstellt, welches beim starten ein minimales Fenster mit **tkinter** öffnet, in welchem man dann eine Stückzahl eingeben kann. Das GUI baut auf dem vorherigen Kompetenznachweis auf, wurde aber bewusst einfach gehalten. Die Eingabe wird dann in einer Queue gespeichert, welche von zwei Worker-Threads im Hintergrund abgearbeitet wird. In der Konsole wird jeweils die Aktivität geloggt:

```
[!] Worker 1 started with package of size 5
[+] Package of size 3 added
[!] Worker 2 started with package of size 3
[-] Worker 1 finished package of size 5
[+] Package of size 2 added
[!] Worker 1 started with package of size 2
[-] Worker 2 finished package of size 3
[-] Worker 1 finished package of size 2
[+] Package of size 5 added
[!] Worker 2 started with package of size 5
[+] Package of size 3 added
[!] Worker 1 started with package of size 3
[+] Package of size 3 added
[-] Worker 2 finished package of size 5
[!] Worker 2 started with package of size 3
[-] Worker 1 finished package of size 3
[+] Package of size 2 added
[!] Worker 1 started with package of size 2
[+] Package of size 1 added
[-] Worker 1 finished package of size 2
[!] Worker 1 started with package of size 1
[-] Worker 2 finished package of size 3
[-] Worker 1 finished package of size 1
[+] Package of size 5 added
[!] Worker 2 started with package of size 5
[-] Worker 2 finished package of size 5
```




```python
import threading
import time
import queue
from tkinter import *

def addPackage():
    print('[+] Package of size {0} added'.format(textbox.get("1.0",'end-1c')))
    waitingQueue.put(textbox.get("1.0",'end-1c'))

def worker(number, waitingQueue):
    while (True):
        productCount = waitingQueue.get()
        print('[!] Worker {0} started with package of size {1}'
            .format(number, str(productCount)))
        time.sleep(int(productCount) / 2)
        print('[-] Worker {0} finished package of size {1}'
            .format(number, str(productCount)))

waitingQueue = queue.Queue()
threading.Thread(target=worker, args=("1", waitingQueue), daemon=True).start()
threading.Thread(target=worker, args=("2", waitingQueue), daemon=True).start()

window = Tk()
window.geometry("200x110")
window.title("Packaging")

label = Label(window, text="Enter numbers to package:")
label.place(x=10,y=10)

textbox = Text(window, height=1, width=20)
textbox.place(x=15,y=40)

button = Button(window, text="Pack", width=22, command=addPackage)
button.place(x=15,y=70)
```