03_Assignment_MayaGrapengießer

December 20, 2022

1 Assignment 03

Python Basics III - Functions and Classes This tutorial was written by Terry L. Ruas (University of Göttingen). The references for external contributors for which this material was anyhow adapted/inspired are in the Acknowledgments section (end of the document).

This notebook will cover the following tasks:

1. Dictionary
2. Classes

1.1 Task 01 – Dictionary

Imagine you have to write a (very simple) bookkeepingsystem for a bank that keeps track of the account balances of each of its customers. 1. Write a function that spans a dictionary holding a default balance of 0 for an initial list of customers. For simplicity, assume customer names are unique identifier. (optional) Can you express that same functionality using a lambda function? 2. What are elegant ways to add or remove single and multiple customers using the functionality of dict? 3. Now write two simple functions that allow you to deposit and withdraw money for a given bank customer. 4. Include error messages for inputs that are not permissible, e.g., withdrawing negative amounts or overdrawing the account, etc.

[8]:
```
customer = {'Leonie': 0, 'Moritz': 0, 'Jessica': 0}

print(customer['Moritz'])
print(customer['Leonie'])
print(customer['Jessica'])
```

```
0
0
0
```

[14]:
```
customer = {'Leonie': 0, 'Moritz': 0, 'Jessica': 0}

del customer['Moritz']
print(customer)

#Alle Löschen
```

```
customer.clear()
print(customer)

customers = {'Tim': 0, 'Maya': 0}
print(customers)
```

```
{'Leonie': 0, 'Jessica': 0}
{}
{'Tim': 0, 'Maya': 0}
```

[16]:
```
def deposit(customer, payment):
    if payment < 0:
        print("Sie können kein negatives Geld einzahlen.")
    else:
        customers[customer] += payment


def withdraw(customer, withdrawal):
    if withdrawal < 0:
        print("Sie können kein negatoves Geld abheben.")
    elif customers[customer]<withdrawal:
        print("Ihr Kontostand ist nicht hoch genug für diese Auszahlung")
    else:
        customers[customer] -= withdrawal

deposit('Tim', -1)
print(customers)
deposit('Tim', 5)
print(customers)
withdraw('Tim', 2.5)
print(customers)
withdraw('Tim', 100)
print(customers)
withdraw('Tim', -5)
print(customers)
```

```
Sie können kein negatives Geld einzahlen.
{'Tim': 2.5, 'Maya': 0}
{'Tim': 7.5, 'Maya': 0}
{'Tim': 5.0, 'Maya': 0}
Ihr Kontostand ist nicht hoch genug für diese Auszahlung
{'Tim': 5.0, 'Maya': 0}
Sie können kein negatoves Geld abheben.
{'Tim': 5.0, 'Maya': 0}
```

1.2 Task 02 – Classes

The manager thinks that the simple bookkeeping system you have built is not powerful enough. She requests that you start from scratch and use classes instead. 1. Write a simple class with appropriate constructor *__init__* that initializes an object of class *Customer* tracking the same information as in Task 01. 2. Now write two simple methods for class *Customer* that allow you to deposit and withdraw money for a given customer object. 3. Include error messages for inputs that are not permissible, e.g., withdrawing negative amounts or overdrawing the account. 4. (Inheritance) Write a child class *SavingsCustomer* that inherits its features from the parent class *Customer*. A savings customer has an extra savings balance for receiving extra interest. The class should have a method to transfer money back and forth between the accounts' main balance as well as the savings balance. Do not forget to add reasonable error messages.

[37]:
```
class Customer:

 def __init__(self, name):

    self.name = name
    self.balance = 0

 def show(self):
        print (f"Balance of {self.name}: {self.balance}")
```

[39]:
```
def __init__(self):
    self.balance = 0
    print ("Konto ist erstellt")

def deposit(self):
    amount = float(input("Enter the amount to be deposit: "))
    self.balance = self.balance + amount
    print ("Kaution ist erfolgreich eingezahlt und die Balance vom Konto ist␣
 ↪%f" %self.balance)

def withdraw(self):
    amount = float(input("Geben Sie den abzuhebenden Betrag an: "))
    if (self.balance >= amount):
        self.balance = self.balance - amount
        print ("Der Rückzug war erfolgreich und die Balance is %" % self.
 ↪balance)
    else:
        print ("Unzureichende Balance")
```

[48]:
```
class SavingsCustomer(Customer):
    def __init__(self, name):
        super().__init__(name)
        self.savings_balance = 0
```

```
    def deposit_savings(self, amount):
        if amount > self.balance:
            print("Der Betrag auf dem Konto ist nicht hoch genug für eine␣
 ↪Auszahlung.")
        else:
            self.savings_balance += amount
            self.balance -= amount


    def withdraw_savings(self, amount):
        if amount < 0:
            print("Du kannst kein Negatives Geld einzahlen.")
        elif self.savings_balance < amount:
            print("Dein Konto ist nicht genug gedeckt für den Antrag der␣
 ↪Auszahlung.")
        else:
            self.savings_balance -= amount
            self.balance += amount

    def show_savings(self):
        print(f"Gespartes Geld von {self.name}: {self.savings_balance}")


c1 = SavingsCustomer("Tim")

c1.show ()
c1.show_savings()
c1.deposit_savings(45)
c1.show()
c1.show_savings()
c1.withdraw_savings(50)
```

```
Gespartes von Tim: 0
Gespartes Geld von Tim: 0
Der Betrag auf dem Konto ist nicht hoch genug für eine Auszahlung.
Gespartes von Tim: 0
Gespartes Geld von Tim: 0
Dein Konto ist nicht genug gedeckt für den Antrag der Auszahlung.
```