03_Assignment_Python

January 10, 2023

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.

[2]:
```
customer = {"Leonie": {"Age" {"20 years old"}, "Town": {"lived in Geesthacht,␣
 ↪now in Göttingen"}, "Bank": {"20€"}}
            "Maya": {"Age": {"20 years old"}, "Town": {"lives in Göttingen"},␣
 ↪"Bank": {"45€"}}
            "Bennet": {"Age": {"18 years old"}, "Town": {"lives in␣
 ↪Geesthacht"}, "Bank": {"100€"}}}
print(customer["Leonie"]["Bank"])

value= customer.pop("Maya")
value

value= customer.pop("Timm": {"Age": {"22 years old"}, "Town": {"lives in␣
 ↪Geesthacht"}})
```

```
  Cell In [2], line 1
    customer = {"Leonie": {"Age" {"20 years old"}, "Town": {"lived in␣
 ↪Geesthacht, now in Göttingen"}, "Bank": {"20€"}}
```

```
                                  ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

[41]:
```
customer = {"Maya": {"Bank": {"20€"}}}

value = customer
value
```

[41]:
```
{'Maya': {'Bank': {'20€'}}}
```

[15]:
```
class Bank:
    def __init__(self):
        self.balance = 0
        print("The account is created")

    def deposit(self):
        amount = float(input("Enter the amount to be deposit: "))
        self.balance = self.balance + amount
        print("Deposit is succesful and the balance in the account is %f" %␣
 ↪self.balance)

    def withdraw(self):
        amount = float(input("Enter the amount to withdraw: "))
        if (self.balance >= amount):
            self.balance = self.balance - amount
            print("The withdraw is successful and balance is %f" % self.balance)
        else:
            print("Not possible. Insuficiet Balance")

    def check(self):
        print("Balance in the account is %f" % self.balance)

acc = Bank()
acc.deposit()
acc.withdraw()
acc.check()
acc.withdraw()
```

```
The account is created
Enter the amount to be deposit:  90
Deposit is succesful and the balance in the account is 90.000000
Enter the amount to withdraw:  100
Not possible. Insuficiet Balance
Balance in the account is 90.000000
```

```
Enter the amount to withdraw:  80

The withdraw is successful and balance is 10.000000
```

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.

[48]:
```python
class Child:
    name = ""
    amount = 0

    def __init__ (self, name, amount):
        self.name = name
        self.amount = amount

    def disposit (self, money):
        if money < 0:
            print("Not possible")
        else:
            self.amount = self.amount + money

    def withdraw (self, money):
        if money < 0:
            print("Not possible")
        elif self.amount - money < 0:
            print("Not possible. Not enough money on acc")
        else:
            self.amount = self.amount - money

Child1 = Child.amount(0)
Child1.disposit2(60)
Child1.withdraw2(70)
print(Child1.amount)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [48], line 23
     20         else:
```

```
     21             self.amount = self.amount - money
---> 23 Child1 = Child.amount(0)
     24 Child1.disposit2(60)
     25 Child1.withdraw2(70)

TypeError: 'int' object is not callable
```

[51]:
```
class save(customer):
    save = 0

    def send (self, money, where):
        if money < 0:
            print("Not possible")

        if where == "h":
            if money > self.amount:
                print("Not possible")
            else:
                self.amount -= money
                self.save += money

        if where == "n":
            if money > self.save:
                print("Not possible")
            else:
                self.save -= money
                self.amount += money


customer2 = save("customer", 15)
customer2.send(5, 'h')

print( "Acc: ",customer2.save)
print("old Acc: ",customer2.amount)

customer2.send(3, 'n')
print("Acc: ",customer2.save)
print("old Acc: ",customer2.amount)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [51], line 1
----> 1 class save(customer):
      2     save = 0
      4     def send (self, money, where):
```

```
TypeError: dict expected at most 1 argument, got 3
```

[ ]: