**Lab 9 - Classes**

**Objective:**

To practice, understand and design Python classes.

**Questions:**

**1. Try to answer this question before running the code. You might run it to check your answer.**

**class NewClass(object):**

**def \_\_init\_\_(self, param\_int=1):**

**self.the\_int = param\_int**

**if param\_int % 2 == 0:**

**self.parity = 'even'**

**else:**

**self.parity = 'odd'**

**def process(self, instance):**

**sum\_int = self.the\_int + instance.the\_int**

**if sum\_int < 0:**

**return 'negative'**

**elif sum\_int % 2 == 0:**

**return 'even'**

**else:**

**return 'odd'**

**def \_\_str\_\_(self):**

**return 'Value {} is {}'.format(self.the\_int, self.parity)**

**inst1 = NewClass(4)**

**inst2 = NewClass(-5)**

**inst3 = NewClass()**

**print(inst1) *# Line 1***

**print(inst1.parity) *# Line 2***

**print(inst1.process(inst2)) *# Line 3***

**print(inst3.process(inst1)) *# Line 4***

**(a) What output is produced by Line 1 of the example program?**

**(b) What output is produced by Line 2 of the example program?**

**(c) What output is produced by Line 3 of the example program?**

**(d) What output is produced by Line 4 of the example program?**

(a) The output produced by Line 1 of the example program is: Value 4 is even. This is because the \_\_str\_\_ method of the NewClass class is called when the print function is used on an instance of the class, and this method returns a string that includes the value of the attribute the\_int and the value of the attribute parity of the instance.

(b) The output produced by Line 2 of the example program is: even. This is because the attribute parity of the inst1 instance is set to 'even' in the constructor of the NewClass class, since the value of param\_int is even.

(c) The output produced by Line 3 of the example program is: negative. This is because the process method of the NewClass class takes another instance of the same class as an argument, and returns the string 'negative' if the sum of the values of the attributes the\_int of the two instances is negative.

(d) The output produced by Line 4 of the example program is: even. This is because the process method of the NewClass class returns the string 'even' if the sum of the values of the attributes the\_int of the two instances is even, and in this case inst1.the\_int + inst3.the\_int equals 4, which is even.

**2. Sample class describing a Person**

|  |
| --- |
| **class Person():**  **def \_\_init\_\_(self, fname, sname, address):**  **self.f\_name = fname**  **self.s\_name = sname**  **self.address = address**  **def change\_address(self, new\_address):**  **self.address = new\_address**  **def \_\_str\_\_(self):**  **return self.f\_name + " "+ self.s\_name + " lives at " + self.address**    ***# main***  **p1 = Person("John", "Smith", "1 Pinebrook street")**  **print(p1.f\_name)**  **print(p1.s\_name)**  **print(p1.address)**    **p1.change\_address("5 Cottage Avenue")**  **print(p1)** |

**(a): Design a class to represent a rectangle. Some methods examples can be the rectangle area and rectangle perimeter.**

**(b): Design a class to represent a bank account. Some information you might want in a bank account are the IBAN, account number, available funds, a list with the last 5 transactions. You might also add methods to withdraw and deposit money.**

class BankAccount(object):

def \_\_init\_\_(self, IBAN: str, account\_number: int, available\_funds=0):

self.IBAN = IBAN

self.account\_number = account\_number

self.available\_funds = available\_funds

self.transactions = []

def deposit(self, amount: float):

if type(amount) != float and type(amount) != int:

print("Please pass a positive value for deposit")

return

if amount < 0:

print("Deposit can only be positive")

return

self.available\_funds += amount

# if len(self.transactions) == 5:

# del self.transactions[0]

self.transactions.append("Deposit of {}".format(amount))

def withdraw(self, amount: float):

if type(amount) != float and type(amount) != int:

print("Please pass a positive value for withdraw")

return

if amount < 0:

print("Withdraw can only be positive")

return

if self.available\_funds - amount < 0:

print("Not enough funds for withdraw")

return

self.available\_funds -= amount

# if len(self.transactions) == 5:

# del self.transactions[0]

self.transactions.append("Withdraw of {}".format(amount))

def \_\_str\_\_(self):

account\_info = "IBAN: " + self.IBAN + "\n"

account\_info += "Account number: " + str(self.account\_number) + "\n"

account\_info += "Available funds: " + str(self.available\_funds) + "\n\n"

account\_info += "Transactions: " + "\n"

if len(self.transactions) >= 5:

for i in range(-5, 0):

account\_info += str(self.transactions[i]) + "\n"

else:

for transaction in self.transactions:

account\_info += transaction

return account\_info

# Main scope

lucas\_account = BankAccount("IE1234567", 1234567)

lucas\_account.deposit(100)

lucas\_account.withdraw(50)

lucas\_account.withdraw(25)

lucas\_account.deposit(101)

lucas\_account.deposit(110)

lucas\_account.withdraw(10)

print(lucas\_account)