Lab 9 - Classes

**Objective:**

To practice, understand and design Python classes.

## Conceptual Practice

**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?

**Value 4 is even**

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

**even**

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

**negative**

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

**odd**

## Critical Thinking Tasks

**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.

**class Rectangle():**

**def \_\_init\_\_(self, length, width):**

**self.length = length**

**self.width = width**

**def area(self):**

**return self.length \* self.width**

**def perimeter(self):**

**return 2 \* (self.length + self.width)**

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

**return f"Rectangle(width={self.width}, length={self.length})"**

**# main()**

**r1 = Rectangle(4, 3)**

**print(r1, "\n")**

**print("Width:", r1.width)**

**print("Length:", r1.length)**

**print("Area:", r1.area())**

**print("Perimeter:", r1.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:**

**def \_\_init\_\_(self, iban, accNo, balance=0):**

**"""Initializes the bank account with IBAN, account number, and an initial balance."""**

**self.iban = iban**

**self.accNo = accNo**

**self.balance = balance**

**self.transactions = []  # To store the last 5 transactions**

**def deposit(self, amount):**

**"""Deposits money into the account and logs the transaction."""**

**if amount > 0:**

**self.balance += amount**

**self.transactions.append(f"Deposited: {amount}")**

**self.\_log\_transaction()**

**else:**

**print("Deposit amount must be positive.")**

**def withdraw(self, amount):**

**"""Withdraws money from the account and logs the transaction if sufficient funds are available."""**

**if amount > 0:**

**if amount <= self.balance:**

**self.balance -= amount**

**self.transactions.append(f"Withdrew: {amount}")**

**self.\_log\_transaction()**

**else:**

**print("Insufficient funds.")**

**else:**

**print("Withdrawal amount must be positive.")**

**def \_log\_transaction(self):**

**"""Keeps only the last 5 transactions in the transaction history."""**

**if len(self.transactions) > 5:**

**self.transactions.pop(0)**

**def get\_transactions(self):**

**"""Returns the list of the last 5 transactions."""**

**return self.transactions**

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

**"""Returns a string representation of the bank account."""**

**return f"Bank Account {self.accNo} (IBAN: {self.iban})\n" \**

**f"Balance: {self.balance}\n" \**

**f"Last Transactions: {', '.join(self.transactions)}"**

**# main()**

**acc1 = BankAccount("IEPBS32435354353", 3745325, 200)**

**print(acc1, "\n")**

**acc1.deposit(200)**

**acc1.withdraw(50)**

**print(acc1, "\n")**

**4.** Write a Shopping Cart class to implement a shopping cart that you often find on

websites where you could purchase some goods. Think about what things you could store in a cart and also what operations you could perform on the cart.

**class ShoppingCart:**

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

**"""Initializes an empty shopping cart."""**

**self.items = {}**

**def add\_item(self, name, price, quantity=1):**

**"""Adds an item to the cart or updates its quantity if already present."""**

**if name in self.items:**

**self.items[name]['quantity'] += quantity**

**else:**

**self.items[name] = {'price': price, 'quantity': quantity}**

**def remove\_item(self, name):**

**"""Removes an item from the cart."""**

**if name in self.items:**

**del self.items[name]**

**else:**

**print(f"Item '{name}' not found in the cart.")**

**def get\_total(self):**

**"""Calculates the total cost of items in the cart."""**

**total = 0  # Initialize total cost to 0**

**for item in self.items.values():  # Iterate over all items in the cart**

**total += item['price'] \* item['quantity']  # Add the cost of the current item to total**

**return total  # Return the calculated total**

**def view\_cart(self):**

**"""Displays the cart items and their total."""**

**if not self.items:**

**print("Your cart is empty.")**

**else:**

**for name, item in self.items.items():**

**print(f"{name}: {item['quantity']} x ${item['price']} = ${item['quantity'] \* item['price']}")**

**print(f"Total: ${self.get\_total()}")**

**# Example usage:**

**cart = ShoppingCart()**

**cart.add\_item("Laptop", 1000, 1)**

**cart.add\_item("Smartphone", 500, 2)**

**cart.view\_cart()**

**cart.add\_item("Laptop", 1000, 1)  # Add another laptop**

**cart.remove\_item("Smartphone")   # Remove smartphone**

**cart.view\_cart()**

**5.** Create a class named Attendee which will keep track of the number of people joining and leaving a meeting. It should have methods to increase the number of people in the meeting if a new attendee joins, and to decrease the number of people in the meeting if someone leaves. Write a program that will display a welcome message and the number of people currently in the meeting every time a new Attendee object is created.

**class Attendee:**

**# Class variable to track the total number of people in the meeting**

**people\_in\_meeting = 0**

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

**"""Increases the number of people in the meeting when a new Attendee is created."""**

**Attendee.people\_in\_meeting += 1  # A new person joins the meeting**

**print(f"Welcome! There are {Attendee.people\_in\_meeting} people in the meeting.")**

**def leave\_meeting(self):**

**"""Decreases the number of people in the meeting when this Attendee leaves."""**

**if Attendee.people\_in\_meeting > 0:**

**Attendee.people\_in\_meeting -= 1**

**print(f"Someone left the meeting. There are now {Attendee.people\_in\_meeting} people in the meeting.")**

**else:**

**print("No one is in the meeting to leave.")**

**def get\_people\_in\_meeting(self):**

**"""Returns the current number of people in the meeting."""**

**return Attendee.people\_in\_meeting**

**# Example usage:**

**attendee1 = Attendee()  # First attendee joins**

**attendee2 = Attendee()  # Second attendee joins**

**attendee3 = Attendee()  # Third attendee joins**

**# Someone leaves the meeting**

**attendee2.leave\_meeting()**

**# Display current number of people**

**print(f"Current people in meeting: {attendee1.get\_people\_in\_meeting()}")**

**6.   
(a)** Create a WizCoinPurse class, which represents a number of coins in a fictional wizard currency contained in a wizard’s purse. In this currency, the denominations are knuts, sickles (worth 29 knuts), and galleons (worth 17 sickles or 493 knuts). Make sure you add a \_\_str\_\_ method.

**class WizCoinPurse:**

**def \_\_init\_\_(self, galleons=0, sickles=0, knuts=0):**

**"""Initializes a WizCoinPurse with the given number of galleons, sickles, and knuts."""**

**self.galleons = galleons**

**self.sickles = sickles**

**self.knuts = knuts**

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

**"""Returns a string representation of the WizCoinPurse."""**

**return f"{self.galleons} Galleons, {self.sickles} Sickles, {self.knuts} Knuts"**

**(b)** Create a value method that returns an integer representing the amount of knuts contained in a purse.

**def knuts\_value(self):**

**"""Returns the total value of the purse in knuts."""**

**total\_knuts = self.knuts + (self.sickles \* 29) + (self.galleons \* 493)**

**return total\_knuts**

**(c)** Create a method that returns the weight of the purse in grams (a float number), knowing that 1 galleon weights 31.103 grams, 1 sickle weights 11.34 grams, and 1 knut weights 5 grams.

**def weight(self):**

**"""Returns the weight of the purse in grams."""**

**total\_weight = (self.galleons \* 31.103) + (self.sickles \* 11.34) + (self.knuts \* 5)**

**return total\_weight**