1. Inheritance - Quiz

Q1 of 3

What is the output of the following code snippet?

```
class Parent:
    def __init__(self,num):
        self.__num=num
    def get_num(self):
        return self.__num
class Child(Parent):
    def show(self):
        print("This is in child class")
son=Child(100)
print(son.get_num())
son.show()
```

- (●) 100 This is in child class ✓
- ( ) Error: Child object, obj does not have __num as an attibute
- ( ) Error: Child class should have a constructor
- ( ) Error: Child object cannot invoke a parent method

Q2 of 3

What is the output of the following code snippet?

```
class Parent:
    def __init__(self,num):
        self.__num=num
    def get_num(self):
        return self.__num
class Child(Parent):
    def __init__(self,val,num):
        self.__val=val
    def get_val(self):
        return self.__val
son=Child(100,10)
print("Parent: Num:",son.get_num())
print("Child: Val:",son.get_val())
```

- (●) Error: child object has no attribute as __num ✓
- ( ) 10 100
- ( ) 100 10

2. Super() - Quiz

Q1 of 7

What is the output of the following code snippet?

```
class Parent:
    def __init__(self,num):
      self.__num=num
    def get_num(self):
      return self.__num
class Child(Parent):
    def __init__(self,num,val):
      super().__init__(num)
      self.__val=val
    def get_val(self):
      return self.__val
son=Child(100,200)
print(son.get_num())
print(son.get_val())
```

- (●) 100 200 ✓
- ( ) 200 100
- ( ) None 200
- ( ) Error: arguments cannot be passed through super() to the parent class constructor

Q2 of 7

What is the output of the following code snippet?

```
class Parent:
    def __init__(self):
        self.num=100

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.var=200
    def show(self):
        print(self.num)
        print(self.var)

son=Child()
son.show()
```

- ○ None 200
- ○ Error: A parent class instance variable cannot be accessed in a child class method
- ◉ 100 200 ✓
- ○ Error: super() can invoke only parameterized constructor of a parent class

Q3 of 7

Consider the following python function for representing the customers of a retail store.5 min
Objective of the code is to record the details of the customers.

```
def customer_record(customer_type, name, discount, points_earned, membership_card_type):
    if(customer_type=="Regular"):
        record="Record Regular Customer:"+name+" "+(str)(discount)
    elif(customer_type=="Privileged"):
        record="Record Privileged Customer:"+name+" "+(str)(points_earned)
    elif(customer_type=="Elite"):
        record="Record Elite Customer:"+name+" "+membership_card_type
    print(record)
```

What will be the optimal class structure if this has to be re-written in Object oriented programming?

- ○ 3 independent classes: Regular, Privileged, Elite
- ○ 1 class: Customer
- ○ 4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged; Grand Child of Privileged: Elite
- ◉ 4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged, Elite ✓
- ○ 4 classes with inheritance: Base class: Customer; Child classes: Regular, Privileged; Grand Child of Regular: Elite

Q4 of 7

What is the output of the following code snippet?

```
class Parent:
    def __init__(self):
        self.__num=100

    def show(self):
        print("Parent:",self.__num)

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.__var=10

    def show(self):
        print("Child:",self.__var)

dad=Parent()
dad.show()
son=Child()
son.show()
```

a) Child: 10
Child: 10

b) Parent: 100
Parent: 100

c) Error: Methods in parent and child classes cannot be same

d) Parent: 100
Child: 10

- ○ a
- ○ b
- ○ c
- ◉ d ✓

Q5 of 7

What should be written in line 14 to get the output as mentioned below?4 min

Parent: 100

Child: 10

```
class Parent:
    def __init__(self):
        self.__num=100

    def show(self):
        print("Parent:",self.__num)

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.__var=10

    def show(self):
        ________________
        print("Child:",self.__var)

obj=Child()
obj.show()
```

- ○ show()
- ◉ super().show() ✓
- ○ self.show()
- ○ Parent.show()

```
    def __init__(self):
        super().__init__()
        self.__overdraft_limit = None

    def get_overdraft_limit(self):
        return self.__overdraft_limit

    def set_overdraft_limit(self, value):
        self.__overdraft_limit = value

    def get_cust_id(self):
        print("Privileged Customer")

class RegularCustomer(Customer):
    def __init__(self):
        super().__init__()
        self.__min_balance = None

    def get_min_balance(self):
        return self.__min_balance

    def set_min_balance(self, value):
        self.__min_balance = value

r=RegularCustomer()
p=PrivilegedCustomer()
r.get_cust_id()
p.get_cust_id()
```

a) Normal Customer
Privileged Customer

b) Privileged Customer
Normal Customer

c) Error

d) None of the above

- (●) a ✓
- ( ) b
- ( ) c
- ( ) d

```
r.get_cust_id()
p.get_cust_id()
n.get_cust_id()
```

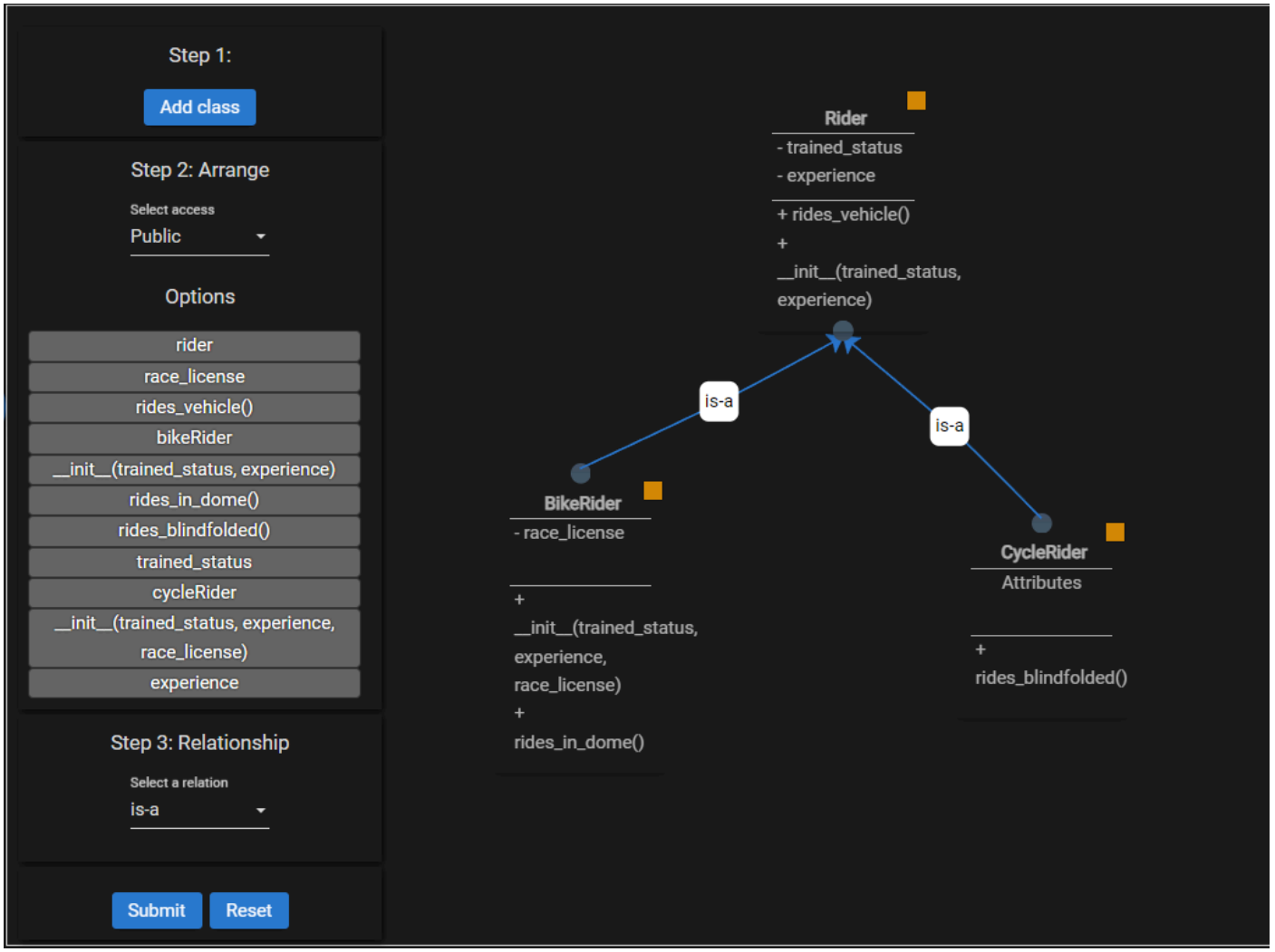
Option B

```
class PrivilegedCustomer(Customer):
    def __init__(self):
        super().__init__()
        self.__overdraft_limit = None

    def get_overdraft_limit(self):
        return self.__overdraft_limit

    def set_overdraft_limit(self, value):
        self.__overdraft_limit = value

    def get_cust_id(self):
        print("Privileged Customer")

class RegularCustomer(Customer):
    def __init__(self):
        super().__init__()
        self.__min_balance = None

    def get_min_balance(self):
        return self.__min_balance

    def set_min_balance(self, value):
        self.__min_balance = value

    def get_cust_id(self):
        print("Regular Customer")

r=RegularCustomer()
p=PrivilegedCustomer()
n=Customer()
r.get_cust_id()
n.get_cust_id()
p.get_cust_id()
```

- ○ A
- ◉ B ✓

3. Exercise on Class Diagram with Inheritance - Level 1

Answer Link -



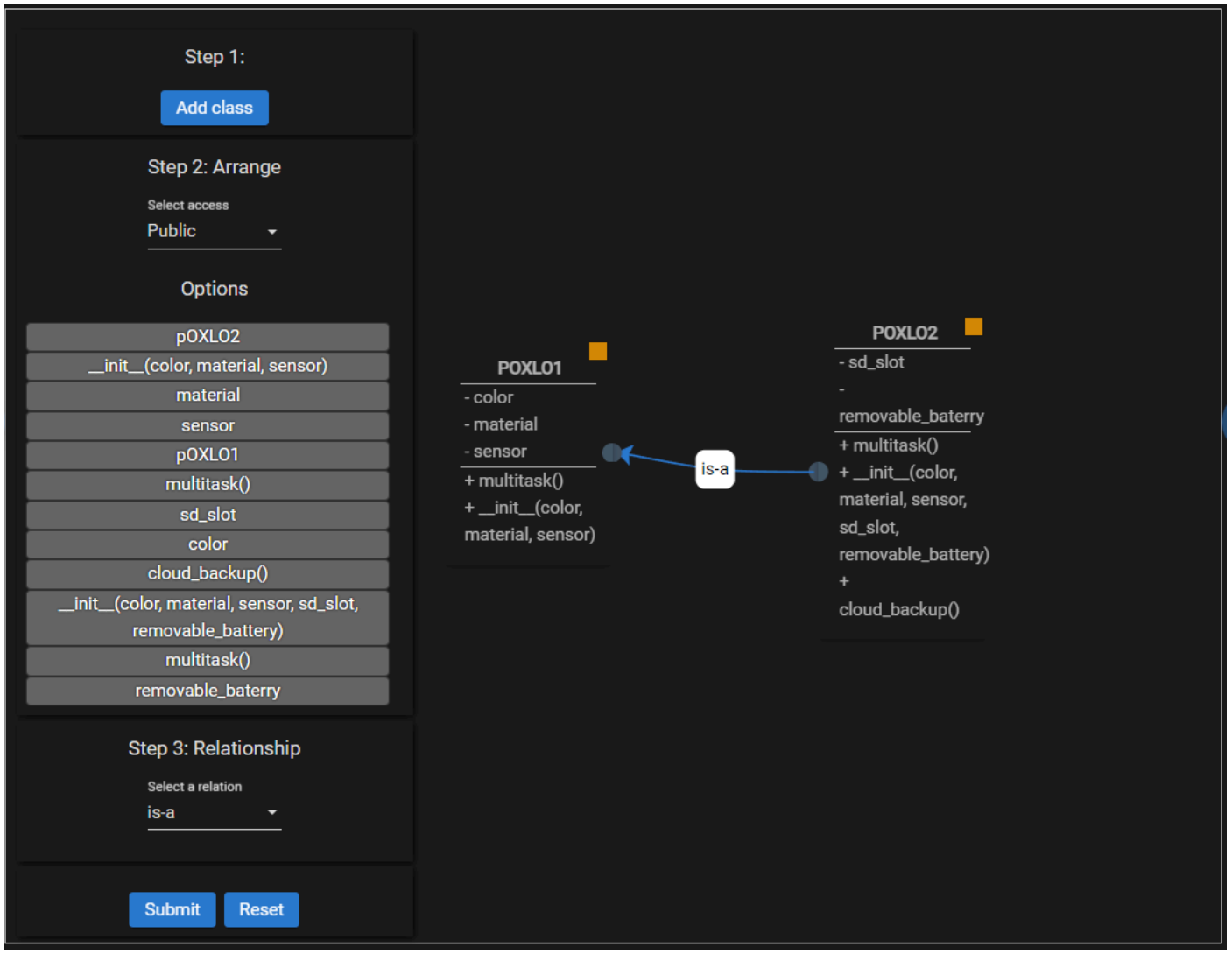
4. Exercise on Inheritance - Level 1

Answer Link -

https://github.com/bsccs2023/Naan-Mudhalvan-2024/blob/d653f38bc86772310bb14c7e14ff9affcbe9f4bc/8--Inheritance/4--Exercise-on-Inheritance-Level-1.py

5. Exercise on Class Diagram with Inheritance - Level 2

Answer Link -



6. Inheritance Types - Quiz

Q1 of 2

What will be the output of the code given below?

```
class A:
    def m1(self):
        return 20
class B(A):
    def m1(self):
        return 30
    def m2(self):
        return 40
class C(B):
    def m2(self):
        return 20
obj1=A()
obj2=B()
obj3=C()
print(obj1.m1() + obj3.m1()+ obj3.m2())
```

- ◯ Error: Method m1 should be overridden in class C.
- ◯ 80
- ◉ 70 ✓
- ◯ 90

Q2 of 2

What will be the output of the code given below?

```
class A:
    def m1(self):
        return 20
class B(A):
    def m1(self):
        val=super().m1()+30
        return val
class C(B):
    def m1(self):
        val=self.m1()+20
        return val
obj=C()
print(obj.m1())
```

- ○ 70
- ○ 20
- ◉ RunTimeError: Maximum recursion depth exceeded ✓