**Lab 11 - Class design and overloading**

**Questions:**

**Q1. Try to answer this question before running the code. You might run it to check your answer.**

**class TestClass(object):**

**def \_\_init\_\_(self,param\_str=''):**

**self.the\_str=''**

**for c in param\_str:**

**if c.isalpha():**

**self.the\_str += c**

**def \_\_add\_\_(self,param):**

**if type(param)==TestClass:**

**the\_str = self.the\_str + param.the\_str # ‘abc’ + ‘ijk’ -> ‘abcijk’**

**return TestClass(the\_str)**

**else:**

**return self**

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

**return 'Value: {}'.format(self.the\_str)**

**inst1 = TestClass('abc') # inst1.the\_str -> ‘abc’**

**inst2 = TestClass('123ijk') # inst2.the\_str -> ‘ijk’**

**sumInst1 = inst1 + inst2 # inst1.\_\_add\_(inst2) -> sumInst1.the\_str -> ‘abcijk’**

**sumInst2 = inst1 + 'xyz' # inst1.\_\_add\_(‘xyz’) -> sumInst2.the\_str -> ‘abc’**

**print(inst1) *# Line 1 # inst1.\_\_str\_\_()***

**print(sumInst1) *# Line 2***

**print(sumInst2) *# Line 3***

**print(isinstance(sumInst2,TestClass)) *# Line 4***

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

Result: abc

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

Result: abcijk

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

Result: abc

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

Result: True

**Q2. Write a class called WholeNumber class. The whole numbers are the non-negative integers: 0,1,2, . . . Your class must handle addition, subtraction, and multiplication of whole numbers—no division or mixed-type (whole number and integer) operations need be handled. Your class must also handle printing—e.g., if x is an instance of the WholeNumber class, you must be able to print x.**

**Two cases must not be allowed:**

**(1) you must not be able to create a WholeNumber that has a negative value;**

**(2) an arithmetic operation cannot be allowed to have a negative result.**

**In both cases, an error message must be printed. Remember that arithmetic must return a whole number. That is, if x and y are whole numbers, the result of x + y must be a whole number. Include sample code that uses your class and demonstrates the use of all methods as well as error handling.**

class WholeNumber:

def \_\_init\_\_(self, value):

if value < 0:

print("Error: WholeNumber must be non-negative.")

else:

self.value = value

def \_\_add\_\_(self, other):

if isinstance(other, WholeNumber) and other.value >= 0:

return WholeNumber(self.value + other.value)

else:

print("Error: Cannot add WholeNumber with negative value.")

return None

def \_\_sub\_\_(self, other):

if isinstance(other, WholeNumber) and other.value <= self.value:

return WholeNumber(self.value - other.value)

else:

print("Error: Subtraction would result in negative WholeNumber.")

return None

def \_\_mul\_\_(self, other):

if isinstance(other, WholeNumber) and other.value >= 0:

return WholeNumber(self.value \* other.value)

else:

print("Error: Cannot multiply by negative WholeNumber.")

return None

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

return str(self.value)

Here's an example of how to use the WholeNumber class:

a = WholeNumber(2)

b = WholeNumber(3)

print(a) # prints "2"

print(b) # prints "3"

c = a + b # c.value is 5

print(c) # prints "5"

d = a - b # prints "Error: Subtraction would result in negative WholeNumber."

print(d) # prints "None"

e = b - a # e.value is 1

print(e) # prints "1"

f = a \* b # f.value is 6

print(f) # prints "6"

g = WholeNumber(-1) # prints "Error: WholeNumber must be non-negative."

**Q3. Write a class for linear equations. A generic linear equation is of the form y = mx + b where m and b are constants. Include the following methods:**

**(a) \_\_init\_\_, \_\_str\_\_, \_\_repr\_\_.**

**(b) value(x), which returns the value of the equation given x.**

**(c) compose(LinearEquation) that composes two linear equations. That is, if y = ax + b and z = cx + d, then y(z)= (a\*c)x +(a\*d + b) and will be called as y.compose(z). Note that the compose operation is not commutative.**

**(d) add returns the sum of two linear equations. That is, if y = ax + b and z = cx + d, then y + z = (a + c)x + (b + d).**

class LinearEquation:

def \_\_init\_\_(self, m, b):

self.m = m

self.b = b

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

return f"y = {self.m}x + {self.b}"

def \_\_repr\_\_(self):

return f"LinearEquation({self.m}, {self.b})"

def value(self, x):

return self.m \* x + self.b

def compose(self, other):

m = self.m \* other.m

b = self.m \* other.b + self.b

return LinearEquation(m, b)

def \_\_add\_\_(self, other):

m = self.m + other.m

b = self.b + other.b

return LinearEquation(m, b)

Here's an example of how to use the LinearEquation class:

a = LinearEquation(2, 3)

b = LinearEquation(-1, 4)

print(a) # prints "y = 2x + 3"

print(b) # prints "y = -1x + 4"

print(a.value(0)) # prints "3"

print(a.value(1)) # prints "5"

print(b.value(0)) # prints "4"

print(b.value(1)) # prints "3"

c = a.compose(b)

print(c) # prints "y = -2x + 7"

d = a + b

print(d) # prints "y = 1x + 7"