H2 Computing Practical Worksheet Review – T1W5

1a Write the code to:

i. Check for valid ISBN-10 and ISBN-13 numbers
- Note that the above should correspond to a single function that detects the given ISBN type (either 10 or 13) and then returns True is valid or False if not.

ii. Generate n random ISBN-10 or ISBN-13 numbers
- There should be 2 separate functions, 1 generating n random ISBN-10 numbers, and the other generating n random ISBN-13 numbers.

The details on the required check digit algorithms can be found on the following page:
https://en.wikipedia.org/wiki/International_Standard_Book_Number#Check_digits

A. Discuss the following code in terms of its correctness and programming practices.

```
def check_isbn(isbn):
    if "-" in isbn:
        isbn13 = False
        if len(isbn) > 13:
            #temporarily removing the first three number for possible isbn13
            temp = isbn[:3]
            isbn = isbn[4:]
            isbn13 = True
        if len(isbn) > 10:
            #removing "-"
            result = ""
            for char in isbn.split("-"):
                result += char
            isbn = result
            if len(isbn) != 10:
                return False, 1
##                return False
        if isbn13:
            #adding back the first three number of isbn13
            isbn = temp + isbn
    if not isbn.isnumeric():
##        return False
        return False, 3
    else:
        s = 0
        if len(isbn) == 10:
            for i in range(len(isbn) - 1):
                s += int(isbn[i]) * (10 - i)
            s = s % 11
            if s != 0:
                s = 11 - s
##            print(s)
        elif len(isbn) == 13:
            for i in range(0, len(isbn) - 1, 2):
                s += int(isbn[i])
            for j in range(1, len(isbn) - 1, 2):
                s += int(isbn[j]) * 3
            s = s % 10
            if s != 0:
                s = 10 - s
        if s == int(isbn[-1]):
            return True
##        return False
        return False, 4
```

B. Discuss the following code and determine if it following good programming practices.

```
def check_13(number):
      flip = True
      s = 0
      for i in range(12):
            if flip == True:
                  s += int(number[i]) * 1
                  flip = False
            else:
                  s += int(number[i]) * 3
                  flip = True
      s = (10 - s%10)%10
      return s
```

C. Is there a better way to do the following?

```
def gen_isbn10(n):
      for i in range(n):
            isbn = ''
            isbn += str(random.randint(0,999999999))
            while len(isbn) < 9:
                  isbn = '0' + isbn
            isbn += check_10(isbn)
            print(isbn)
```

D. Is the following function correct?

```
def gen_isbn13(n):
      for i in range(n):
            isbn = '978' + str(random.randint(0,999999999))
            while len(isbn) < 12:
                  isbn = '0' + isbn
            isbn += str(check_13(isbn))
            print(isbn)
```

E. Does the following code following good programming practices?

```
def generate_n_isbn_10(n):
       isbn = ""
       for i in range(9):
              isbn = isbn + chr(random.randint(0,9))
              summ += int(isbn[len(isbn)-1]) * (10-i)
       for i in range(9):
              if (summ+i)%11 == i:
                     isbn = isbn + chr(i)
                     return isbn
       isbn = isbn + "X"
       print(isbn)
       return isbn

def generate_n_isbn_13(n):
       isbn = ""
       for i in range(12):
              isbn = isbn + chr(random.randint(0,9))
              summ += int(isbn[len(isbn)-1]) * (12-i)
       for i in range(9):
              if (summ+i)%10 == i:
                     isbn = isbn + chr(i)
                     print(isbn)
                     return isbn
```

F. Does the following code work? Are there any programming practice issues?

```
def check_isbn(isbn):
    if len(isbn) == 10:
        if not isbn.isnumeric():
            return False
        sum_value = 0
        for i in range(9):
            sum_value += int(isbn[i])*(10-i)
        if (11 - (sum_value % 11)) % 11 == int(isbn[9]):
            return True
        elif (11 - (sum_value % 11)) % 11 == 10 and isbn[9] == 'X':
            return True
        else:
            return False
    elif len(isbn) == 13:
        if not isbn.isalnum():
            return False
        sum_value = 0
        for i in range(12):
            try:
                if i % 2 == 0:
                    sum_value += int(isbn[i])
                else:
                    sum_value += int(isbn[i]) * 3
            except ValueError:
                return False
        if 10 - sum_value % 10 == int(isbn[-1]):
            return True
        elif 10 - sum_value % 10 == 10 and str(isbn[-1]) == 0:
            return True
        else:
            return False
    else:
        return False
```

G. Take note of the following implementation (which does not use format). How does the % operator work under the following context in terms of formatting?

```
def gen_isbn_10(n):
    for i in range(n):
        isbn = '%09d' % random.randint(0, 999999999)
        check_digit = check_isbn_10(isbn)
        yield isbn + check_digit

def gen_isbn_13(n):
    for i in range(n):
        isbn = '%012d' % random.randint(0, 999999999999)
        check_digit = check_isbn_13(isbn)
        yield isbn + check_digit
```

H. Consider the following line of code. Do you know what it does?

```
to_mod += (1 + 2*(i&1))*int(isbn[i])
```

I. Can you improve on the following?

```
def isbn_check(isbn):
    total = 0
    if len(isbn) == 13:
        multiplier = 1
        turn = 0
        for i in isbn[:12]:
            total += int(i) * multiplier
            turn += 1
            if turn % 2 != 0:
                multiplier = 3
            else:
                multiplier = 1
        checkdigit = 10 - (total % 10)
        if str(checkdigit) == isbn[-1]:
            return True
        else:
            return False
    elif len(isbn) == 10:
        multiplier = 10
        for i in isbn[:9]:
            total += int(i) * multiplier
            multiplier -= 1
        checkdigit = 11 - (total % 11)
        if checkdigit == 10:
            checkdigit = 'X'
        if str(checkdigit) == isbn[-1]:
            return True
        else:
            return False
```

1b Write the object-oriented code for a circular doubly-linked linked list. Your implementation should ensure the following methods:

- Fully object-oriented
- Doubly-linked
- Circular
- Able to store generic data objects
- Insert methods - both at the front and back of the list
- Find methods - returning True if the given data object is stored in the linked list, and False otherwise
- Delete method, which utilises the find method and then removes the given entry if possible; it must return True is successfully removed, or else will return False
- Print method should print the complete contents of the linked list, including any sentinel nodes and all linking data (i.e., there should be some formatting to indicate how the nodes are linked – ensure proper UI principles are adhered to when formatting this output)

A. Discuss the following code in terms of its correctness and programming practices:

```
class Circular_Doubly_Linked_List_Node():
    def __init__(self, data):
        self._data = data
        self._next = None
        self._prev = None

    def get_data(self):
        return self._data

    def set_data(self, new_data):
        self._data = new_data

    def get_next(self):
        return self._next

    def set_next(self, new_next):
        self._next = new_next

    def get_prev(self):
        return self._prev

    def set_prev(self, new_prev):
        self._prev = new_prev

class Circular_Doubly_Linked_List():
    def __init__(self):
        self._root = None
        self._last = None

    def insert(self, data, position = 1):
        # position = 1 for inserting in front
        # position = -1 for inserting behind
        if self._root == None:
            self._root = Circular_Doubly_Linked_List_Node(data)
            self._root.set_prev(self._root)
            self._root.set_next(self._root)
            self._last = self._root
        else:
            self._root.set_prev(Circular_Doubly_Linked_List_Node(data))
            self._last.set_next(self._root.get_prev())
            self._root.get_prev().set_prev(self._last)
            self._root.get_prev().set_next(self._root)
            if position == 1:
                self._root = self._root.get_prev()
            elif position == -1:
                self._last = self._last.get_next()
    def find(self, data):
        if self._find_node(data) == False:
            return False
        else:
            return True
```

```
    def _find_node(self, data):
        if self._root == None:
            return False
        cur = self._root
        if cur.get_data() == data:
            return True, cur
        else:
            while cur != self._last:
                cur = cur.get_next()
                if cur.get_data() == data:
                    return True, cur
            return False

    def delete(self, data):
        try:
            node = self._find_node(data)[1]
        except TypeError:
            print("Data to be deleted is not in list")
        else:
            if node == self._root and node == self._last:
                self._root = None
                self._last = None
            else:
                if node == self._root:
                    self._root = node.get_next()
                elif node == self._last:
                    self._last = node.get_prev()
                node.get_prev().set_next(node.get_next())
                node.get_next().set_prev(node.get_prev())
```

B. Some implementations of the insert methods were done within a single method. This was facilitated by the specification of an addition parameter. Consider the following pieces of code and discuss if this is good practice.

```
    def insert(self, data, position):
        new_node = node(data)
        if self._root.get_prev() == None:
            self._root.set_next(new_node)
            self._root.set_prev(new_node)
            new_node.set_next(self._root)
            new_node.set_prev(self._root)
        else:
            if position == 'back':
                self._root.get_prev().set_next(new_node)
                new_node.set_prev(self._root.get_prev())
                new_node.set_next(self._root)
                self._root.set_prev(new_node)
            else:
                self._root.get_next().set_prev(new_node)
                new_node.set_next(self._root.get_next())
                self._root.set_next(new_node)
                new_node.set_prev(self._root)
```

C. Some other implementations instead utilised two separate methods. Consider the following implementation and determine if they follow good programming practices

```
      def insert_front(self,data):
            if self._root == None:
                  self._root = Node(data)
                  self._root.set_next(self._root)
                  self._root.set_prev(self._root)
            else:
                  temp = self._root
                  self._root = Node(data)
                  self._root.set_next(temp)
                  temp.set_prev(self._root)


      def insert_back(self,data):
            if self._root == None:
                  self._root = Node(data)
                  self._root.set_next(self._root)
                  self._root.set_prev(self._root)
            else:
                  temp = self._root
                  self._root = Node(data)
                  self._root.set_prev(temp)
                  temp.set_next(self._root)
```

D. Are there any issues with the following implementation?

```
class Node():
       def __init__(self,initdata):
              self.__data=initdata
              self.__next=None
        self.__pre=None

       def get_data(self):
              return self.__data

       def get_next(self):
              return self.__next

       def get_pre(self):
              return self.__pre

       def new_data(self,new_d):
              self.__data=new_d

       def new_next(self,new_n):
              self.__next=new_n

       def new_pre(self,new_p):
              self.__pre=new_p

       def __str__(self):
              return str(self.__data)
```

E. Discuss if the prev_node and next_node parameters are necessary in the following.

```
class Sentinel():
    def __init__(self, prev_node=None, next_node=None):
        self._prev = prev_node
        self._next = next_node

    def get_prev(self):
        return self._prev

    def get_next(self):
        return self._next

    def set_prev(self, new_prev):
        self._prev = new_prev

    def set_next(self, new_next):
        self._next = new_next

class DoubleNode(Sentinel):
    def __init__(self, data, prev_node=None, next_node=None):
        super().__init__(prev_node, next_node)
        self._data = data

    def get_data(self):
        return self._data

    def set_data(self, new_data):
        self._data = new_data
```

F. Are there any issues with the following implementation?

```
    def insert_front(self, data):
        if self._root == None:
            self._root = Sentinel()
            node = DoubleNode(data, self._root, self._root)
            self._root.set_next(node)
            self._root.set_prev(node)
        else:
            last_node = self._root.get_prev()
            node = DoubleNode(data, last_node, self._root)
            last_node.set_next(node)
            self._root.set_prev(node)

    def insert_back(self, data):
        if self._root == None:
            self._root = Sentinel()
            node = DoubleNode(data, self._root, self._root)
            self._root.set_next(node)
            self._root.set_prev(node)
        else:
            first_node = self._root.get_next()
            node = DoubleNode(data, self._root, first_node)
            first_node.set_prev(node)
            self._root.set_next(node)
```

1c Write the object-oriented code for a hash table that utilised chaining for conflict resolution. Your implementation should ensure the following methods:

- Fully object-oriented
- Initialisation should be based on a given expected size of the hash table
- For chaining, your buckets should be implemented using the linked list from 1b
- Able to store generic data objects
- Insert method using hash values
- You may use Python's inbuilt hash() method for all hash value calculations
- Find method - returning True if the given data object is stored in the linked list, and False otherwise; again, you must use hash values to perform this
- Delete method, which utilises the find method and then removes the given entry if possible; it must return True is successfully removed, or else will return False
- Print method should utilise the print method of each linked list with the appropriate separator

A. Many of you decided to first populate the Hash Table with None values; the buckets (i.e., lists) would only be initialised when new elements are to be inserted into a bucket. Discuss pre-initialising versus the above described method to determine which should be used.

B. Are there any issues with the following implementation?

```
class HashTable():
        def __init__(self,n):
                self.__hashtable = [LinkedList]*n

        def insert(data):
                k = hash(data)
                self.__hashtable[k].insert(data)

        def find(data):
                k = hash(data)
                return self.__hashtable[k].find(data)

        def delete(data):
                if not find(data):
                        return False
                k = hash(data)
                self.__hashtable[k].delete(data)
                return True

        def __str__(self):
                for i in range(len(self.__hashtable)):
                        self.__hashtable[i].__str__()
```

C. Are there any issues with the following implementation?

```
class HashTable():
    def __init__(self, size):
        self._table = []
        for i in range(size):
            self._table.append(DoublyLinkedList())

    def insert(self, data):
        hash_value = hash(data)
        self._table[hash_value].insert(data)
        return True

    def find(self, data):
        return self._table[hash(data)].find(data)

    def delete(self, data):
        return self._table[hash(data)].delete(data)

    def print(self):
        for i in range(len(self._table)):
            print()
            print("BUCKET "+str(i))
            print()
            self._table[i].print()
```

1d Use your code from parts 1a, 1b and 1c to implement a data entry interface to enter both ISBN-10 and ISBN-13 codes. This interface should be:

- Text-based
- Utilise the validation methods in 1a to ensure ISBN codes entered are valid (or else reject them)
- Utilise the hash table in 1c to store the values

1e Design a set of test cases to evaluate your code in 1d. Your test cases should evaluate all aspects of the data entry interface.

A. Some of you implemented a menu. However, the functionality is questionable. Review the following code to determine if it implements what the question required.

```
def menu():
    initialised = False
    while True: #while the menu has not ended yet (i.e.: 6 not chosen)
        choice = input("\nSelect a number from 1-6:\n1. Start\n2. Insert\n3. Find\n4. Delete\n5. Print\n6. End\n")
        if not choice.isnumeric():
            print("Invalid input type! Enter a digit corresponding to its option!")
            continue
        elif int(choice) not in range(1, 7):
            print("Invalid input range! Enter a digit from 1-6!")
            continue
        elif choice == "1":
            table = HashTable()
            initialised = True
        elif choice == "6":
            break
        else:
            if not initialised:
                print("Select option 1 first!")
                continue
            else:
                if choice == "5":
                    table.print()
                else:
                    get_isbn = True
                    while get_isbn:
                        isbn = input("Enter an ISBN: ('STOP' to stop)")
                        if isbn == "STOP":
                            get_isbn = False
                        if check_isbn(isbn) == False:
                            print("Invalid ISBN!")
                        else:
                            get_isbn = False
                    if choice == "2":
                        table.insert(isbn)
                    elif choice == "3":
                        table.find(isbn)
                    elif choice == "4":
                        table.delete(isbn)
```

B. Some of the submissions included test cases, but not the code that would actually carry out testing using those test cases. Discuss what the code in 1e should do.

Other general programming practice issues:

A. Naming conventions … again