Lab 4: Dictionaries and Classes

Name: _____ University Number: _____

Exercise 1: Canadian Postal Codes

AIM:

In Canada, the first, third, and fifth characters in the postal code are letters while the second, fourth, and sixth characters are digits. We can determine the province or territory in which an address resides from the first character of its postal code as shown in the following table. No valid postal codes currently begin with D, F, I, O, Q, U, W, or Z.

Province/Territory	First Character(s)
Newfoundland and Labrador	A
Nova Scotia	B
Prince Edward Island	C
New Brunswick	E
Quebec	G, H, and J
Ontario	K, L, M, N, and P
Manitoba	R
Saskatchewan	S
Alberta	T
British Columbia	V
Nunavut or Northwest Territories	X
Yukon	Y

The second character in a postal code identifies whether the address is rural or urban. If that character is a 0, then the address is rural; otherwise, it is urban. Write a Python program that prompts a postal code from the user and displays the province or territory associated with it, along with whether the address is urban or rural. Your program should use a dictionary to map from the first character of the postal code to the province or territory name. Here are the sample input and output of the program:

```
Enter a Canadian postal code: X0D 0A0
The postal code is for a rural address in Nunavut or Northwest
Territories.
```

```
Enter a Canadian postal code: H8Z 7R6
The postal code is for an urban address in Quebec.
Enter a Canadian postal code: ABB 2L5
The second character in the postal code must be a digit!
Enter a Canadian postal code: Q5N 8P4
The first character in the postal code is invalid!
It cannot be D, F, I, O, Q, U, W, or Z!
```

ALGORITHM:

1. Start
2. Construct a dictionary for the first characters in the postal code to the province or territory
3. Read a Canadian postal code from the user
4. Display whether the postal code refers to an urban or a rural address along with the associated province or territory
5. End

PROGRAM:

```
# Exercise 1: Canadian Postal Codes
# Written by F K Chow, HKU
# Last update: 2022/2/21

# Construct a dictionary for the first characters in the postal code
# to the province or territory
district = {'A':'Newfoundland and Labrador', 'B':'Nova Scotia',
            'C':'Prince Edward Island', 'E':'New Brunswick',
            'G':'Quebec', 'H':'Quebec', 'J':'Quebec', 'K':'Ontario',
            'L':'Ontario', 'M':'Ontario', 'N':'Ontario',
            'P':'Ontario', 'R':'Manitoba', 'S':'Saskatchewan',
            'T':'Alberta', 'V':'British Columbia',
            'X':'Nunavut or Northwest Territories', 'Y':'Yukon'}

# Read a Canadian postal code from the user
pcode = input('Enter a Canadian Postal Code: ')
```

```
# Display whether the postal code refers to an urban or a rural
# address along with the associated province or territory
ch1 = pcode[0]
if not (ch1 in district):
    print('The first character in the postal code is invalid!')
    print('It cannot be D, F, I, O, Q, U, W, or Z!')
else:
    ch2 = pcode[1]
    if (ord(ch2) < 48) or (ord(ch2) > 57):
        print('The second character in the postal code must be a \
digit!')
    else:
        print('The postal code is', end=' ')
        if ord(ch2) == 48:
            print('a rural address in', end=' ')
        else:
            print('an urban address in', end=' ')
        print('{:s}.'.format(district[ch1]))
```

OUTPUT:

```
Enter a Canadian Postal Code: X1A 0V0
The postal code is an urban address in Nunavut or Northwest Territories.
Enter a Canadian Postal Code: N0H 6S8
The postal code is a rural address in Ontario.
Enter a Canadian Postal Code: D8G 1A0
The first character in the postal code is invalid!
It cannot be D, F, I, O, Q, U, W, or Z!
Enter a Canadian Postal Code: TPS 3C7
The second character in the postal code must be a digit!
```

Exercise 2: Converting Floating-Point Values to Monetary Amounts

AIM:

Write a Python class `MoneyFmt` for converting floating-point values to monetary amounts. The class has one data attribute of type `float` for the floating-point value called `value`. It also provides the following methods:

(a) `__init__(self, value=0.)` for creating a new instance of the class with the floating-point value initialized,

(b) `update(self, value=None)` for updating the floating-point value,

(c) `__str__(self)` for displaying the floating-point value as a monetary amount to two decimal places with the correct sign,

(d) `__repr__(self)` for returning a string such that `eval` applied to the string recreates the instance,

(e) `isnonzero(self)` for checking if the floating-point value is non-zero where a value with absolute value less than 0.005 is regarded as zero.

Here are the sample input and output for using the class `MoneyFmt`:

```
>>> cash = MoneyFmt(135.79)
>>> cash
MoneyFmt(135.79)
>>> print(cash)
$135.79
>>> cash.update(5000000.2468)
>>> cash
MoneyFmt(5000000.2468)
>>> print(cash)
$5,000,000.25
>>> cash.update(-1386.42)
>>> print(cash)
-$1,386.42
>>> cash.isnonzero()
True
>>> cash.update(-0.0049)
>>> cash.isnonzero()
```

```
False
```

ALGORITHM:

1. Start
2. Define the class `MoneyFmt` as follows:
   (a) Define the constructor `__init__` which creates a new instance of the class with the attribute `value` initialized
   (b) Define the method `update` which updates the instance variable `value`
   (c) Define the method `__str__` which displays the instance variable `value` as a monetary amount to two decimal places with the correct sign
   (d) Define the method `__repr__` which returns the string such that applying `eval` on the string recreates the instance
   (e) Define the method `isnonzero` which check if the instance variable `value` is non-zero where a value with absolute value less than 0.005 is regarded as zero
3. End

PROGRAM:

```
# Exercise 2: Converting Floating-Point Values to Monetary Amounts
# Written by F K Chow, HKU
# Last update: 2022/2/21

class MoneyFmt:
    """ A class converting floating-point values to monetary
        amounts """

    def __init__(self, value=0.):
        """ Initialize the floating-point value with the given
            value """
        self.value = value

    def update(self, value=None):
        """ Update the floating-point value """
        self.value = value
```

```
    def __str__(self):
        """ Display the floating-point value as a monetary amount
            to two decimal places with the correct sign """
        if self.value > 0:
            ms = '$'
        else:
            ms = '-$'
        s = '{:.2f}'.format(round(abs(self.value)*100)/100)
        if len(s) > 6:
            n = len(s) - 6
            if n%3 != 0:
                ms += s[0:n%3] + ','
            for i in range(n%3, n, 3):
                ms += s[i:i+3] + ','
            ms += s[-6:]
        else:
            ms += s
        return ms

#Alterative way to define __str__:
#   def __str__(self):
#       """ Display the floating-point value as a monetary amount
#           with the correct sign """
#       if self.value > 0:
#           return '${:,.2f}'.format(self.value)
#       else:
#           return '-${:,.2f}'.format(abs(self.value))

    def __repr__(self):
        """ Return a string such that eval applied to the string
            recreates the instance """
        return "MoneyFmt({0:s})".format(str(self.value))

    def isnonzero(self):
        """ Check if the floating-point value is non-zero where a
```

```
            value with absolute value < 0.005 is regarded as zero """
        return abs(self.value) >= 0.005
```

OUTPUT:

```
>>> cash = MoneyFmt(321321.4567)
>>> cash
MoneyFmt(321321.4567)
>>> print(cash)
$321,321.46
>>> cash.update(-12345678.246)
>>> print(cash)
-$12,345,678.25
>>> cash.update(-0.0064)
>>> print(cash)
-$0.01
>>> cash.isnonzero()
True
>>> cash.update(0.0036)
>>> print(cash)
$0.00
>>> cash.isnonzero()
False
```

Exercise 3: Queue

AIM:

A queue behaves like a real-world queue such as a checkout line in a supermarket. The customer at the front of the line is serviced first and other customers can only enter at the back of the line. So the first customer in line is always the first one to be serviced (i. e. it follows the First In First Out principle). Write a Python class `Queue` for representing queues of integers. The class has two data attributes `qlst` and `maxsize` where `qlst` is the list holding the elements of the queue and `maxsize` is the size limit for the queue. It also provides the following methods:

(a) `__init__(self, maxsize=10)` for initializing the queue with an empty list and a size limit,

(b) `enqueue(self, item)` for adding an element to the back of a non-full queue,

(c) `dequeue(self)` for removing an element from the front of a non-empty queue,

(Hint: use the `pop` function)

(d) `isempty(self)` for checking whether the queue is empty,

(e) `isfull(self)` for checking whether the queue is full (i.e. whether the number of elements in the queue is equal to `maxsize`),

(f) `count(self)` for returning the number of elements in the queue,

(g) `__str__(self)` for displaying all the elements in the queue.

Note that in the method `enqueue`, the provided element would be added to the queue only if it is an integer. Here are the sample input and output for using the class `Queue`:

```
>>> a = Queue(3)
>>> a.isempty()
True
>>> a.enqueue(1.2)
You can only add integers to the queue!
>>> a.enqueue(1)
>>> a.enqueue(2)
>>> a.enqueue(3)
>>> a.enqueue(4)
The queue is full and no new element can be added!
>>> a.isfull()
True
```

```
>>> a.count()
Number of elements in the queue = 3
>>> a.dequeue()
1
>>> print(a)
Element 1 in the queue = 2
Element 2 in the queue = 3
>>> a.dequeue()
2
>>> a.dequeue()
3
>>> a.dequeue()
The queue is empty and there is no front element!
```

ALGORITHM:

1. Start
2. Construct the class `Queue` for representing queues of integers as follows:
   (a) Define the constructor `__init__` which creates a new instance of the class with the attributes `qlst` and `maxsize` initialized
   (b) Define the method `enqueue` which adds an element to the back of the queue
   (c) Define the method `dequeue` which removes an element from the front of the queue
   (d) Define the method `isempty` which checks if the queue is empty
   (e) Define the method `isfull` which checks if the queue is full
   (f) Define the method `count` which counts the number of elements in the queue
   (g) Define the method `__str__` which displays all the elements in the queue
3. End

PROGRAM:

```
# Exercise 3: Queue
# Written by F K Chow, HKU
# Last update: 2022/3/29

class Queue:
```

```
    """ A class representing queues of integers """

    def __init__(self, maxsize=10):
        """ Initialize the queue with an empty list and a size
            limit """
        self.qlst = []
        self.maxsize = maxsize

    def enqueue(self, item):
        """ Add an element to the back of a non-full queue """
        if self.isfull():
            print('The queue is full and no new element can be added!')
        else:
            if isinstance(item, int):
                self.qlst.append(item)
            else:
                print('You can only add integers to the queue!')

    def dequeue(self):
        """ Remove an element from the front of a non-empty queue """
        if self.isempty():
            print('The queue is empty and there is no front element!')
        else:
            print(self.qlst.pop(0))

    def isempty(self):
        """ Check whether the queue is empty """
        return len(self.qlst) == 0

    def isfull(self):
        """ Check whether the queue is full """
        return len(self.qlst) == self.maxsize

    def count(self):
        """ Return the number of elements in the queue """
```

```
        print('Number of elements in the queue =', len(self.qlst))


    def __str__(self):
        """ Print out all the elements in the queue """
        s = ""
        for i in range(len(self.qlst)):
            s += 'Element {0:d} in the queue = '.format(i+1)
            s += '{0:d}\n'.format(self.qlst[i])
        return s[:-1]
```

OUTPUT:

```
>>> a = Queue(5)
>>> a.enqueue(1.3579)
You can only add integers to the queue!
>>> for i in range(1,6):
        a.enqueue(i)


>>> a.enqueue(6)
The queue is full and no new element can be added!
>>> print(a)
Element 1 in the queue = 1
Element 2 in the queue = 2
Element 3 in the queue = 3
Element 4 in the queue = 4
Element 5 in the queue = 5
>>> for i in range(1,6):
        a.dequeue()


1
2
3
4
5
>>> a.isempty()
True
>>> a.dequeue()
The queue is empty and there is no front element!
```

Exercise 4: Quadrilaterals, Trapezoids, Parallelograms, and Squares

AIM:

Suppose a Python class `Point` is defined in the program `point.py` as follows:

```
class Point:
    """ A class representing geometric points """

    def __init__(self, x, y):
        """ Initialize the point with its x, y coordinates """
        self.x = x
        self.y = y

    def getx(self):
        """ Return the x coordinate of the point """
        return self.x

    def gety(self):
        """ Return the y coordinate of the point """
        return self.y

    def __str__(self):
        """ Return a string representation of the point """
        return "({:.1f}, {:.1f})".format(self.x, self.y)
```

Write a Python class `Quadrilateral` for representing quadrilaterals. This class has four data attributes `pt1`, `pt2`, `pt3`, `pt4` of class `Points` for the four endpoints of the quadrilateral. In addition to the constructor for initializing the instance, it provides the methods `getpt1(self)`, `getpt2(self)`, `getpt3(self)`, `getpt4(self)` for returning the individual endpoints of the quadrilateral, `getcoorstr(self)` for returning a string containing the coordinates of the endpoints of the quadrilateral, and `__str__(self)` for returning a string representation of the quadrilateral with the coordinates of its endpoints obtained by calling the method `getcoorstr`. Next, write a Python class `Trapezoid` for representing trapezoids by inheriting from the class `Quadrilateral`. It overrides the method `__str__(self)` for returning a string representation of the trapezoid with the coordinates of its endpoints, its height, and its area. It also adds the methods `getheight(self)` for returning the height of the trapezoid, `getsumoftwosides(self)` for

returning the sum of the length of the two parallel sides of the trapezoid, and `getarea(self)` for returning the area of the trapezoid, assuming the points on each parallel side to have the same *y* coordinates. Next, write a Python class `Parallelogram` for representing parallelograms by inheriting from the class `Trapezoid`. It overrides the method `__str__(self)` for returning a string representation of the parallelogram with the coordinates of its endpoints, its width, its height, and its area. It also adds the methods `getwidth(self)` for returning the width of the parallelogram. Finally, write a Python class `Square` for representing squares by inheriting from the class `Parallelogram`. It overrides the method `__str__(self)` for returning a string representation of the square with the coordinates of its endpoints, its side, and its area. Assume the constructors of these classes are called with valid input. Here are the sample input and output for using these classes:

```
>>> a = Quadrilateral(1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4)
>>> print(a)
Coordinates of the Quadrilateral are:
(1.1, 1.2), (6.6, 2.8), (6.2, 9.9), (2.2, 7.4)
>>> b = Trapezoid(0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0)
>>> b.getsumoftwosides()
14.7
>>> print(b)
Coordinates of the Trapezoid are:
(0.0, 0.0), (10.0, 0.0), (8.0, 5.0), (3.3, 5.0)
Height = 5.0
Area = 36.75
>>> c = Parallelogram(5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0)
>>> print(c)
Coordinates of the Parallelogram are:
(5.0, 5.0), (11.0, 5.0), (12.0, 20.0), (6.0, 20.0)
Width = 6.0
Height = 15.0
Area = 90.00
>>> d = Square(4.0, 0.0, 8.0, 0.0, 8.0, 4.0, 4.0, 4.0)
>>> print(d)
Coordinates of the Square are:
(4.0, 0.0), (8.0, 0.0), (8.0, 4.0), (4.0, 4.0)
Side = 4.0
Area = 16.00
```

ALGORITHM:

1. Start
2. Import the `Point` class and the `isclose` function from the `math` module
3. Construct the class `Quadrilateral` for representing quadrilaterals as follows:
   (h) Define the constructor `__init__` which creates a new instance of the class with the four attributes `pt1`, `pt2`, `pt3`, `pt4` of class `Points` initialized
   (i) Define the method `getpt1`, `getpt2`, `getpt3`, `getpt4` which return the individual endpoints of the quadrilateral
   (j) Define the method `getcoorstr` which returns a string contains the coordinates of the endpoints of the quadrilateral
   (k) Define the method `__str__` which returns a string representation of the quadrilateral
4. Construct the class `Trapezoid` for representing trapezoids by inheriting from the class `Quadrilateral` as follows:
   (a) Define the new method `getheight` which returns the height of the trapezoid
   (b) Define the new method `getsumoftwosides` which returns the sum of the two parallel sides of the trapezoid
   (c) Define the new method `getarea` which returns the area of the trapezoid
   (d) Override the method `__str__` which returns a string representation of the trapezoid
5. Construct the class `Parallelogram` for representing parallelograms by inheriting from the class `Trapezoid` as follows:
   (a) Define the new method `getwidth` which returns the width of the parallelogram
   (b) Override the method `__str__` which returns a string representation of the parallelogram
6. Construct the class `Square` for representing squares by inheriting from the class `Parallelogram` as follows:
   (c) Override the method `__str__` which returns a string representation of the square
7. End

PROGRAM:

```
# Exercise 4: Quadrilaterals, Trapezoids, Parallelograms, and Squares
# Written by F K Chow, HKU
# Last update: 2022/2/21

# Import the Point class and the isclose function from the math module
```

```
from math import isclose
from point import Point

class Quadrilateral:
    """ A class representing quadrilaterals """

    def __init__(self, x1, y1, x2, y2, x3, y3, x4, y4):
        """ Initialize the quadrilateral with its four endpoints """
        self.pt1 = Point(x1, y1)
        self.pt2 = Point(x2, y2)
        self.pt3 = Point(x3, y3)
        self.pt4 = Point(x4, y4)

    def getpt1(self):
        """ Return the first endpoint of the quadrilateral """
        return self.pt1

    def getpt2(self):
        """ Return the second endpoint of the quadrilateral """
        return self.pt2

    def getpt3(self):
        """ Return the third endpoint of the quadrilateral """
        return self.pt3

    def getpt4(self):
        """ Return the fourth endpoint of the quadrilateral """
        return self.pt4

    def getcoorstr(self):
        """ Return a string containing the coordinates of the endpoints
            of the quadrilateral """
        return '{0}, {1}, {2}, {3}'.format(self.pt1, self.pt2,
                                           self.pt3, self.pt4)
```

```
    def __str__(self):
        """ Return a string representation of the quadrilateral """
        return 'Coordinates of the Quadrilateral are:\n'\
               +self.getcoorstr()

class Trapezoid(Quadrilateral):
    """ A class representing trapezoids """

    def getheight(self):
        """ Return the height of the trapezoid """
        if isclose(self.getpt1().gety(), self.getpt2().gety()):
            return abs(self.getpt2().gety() - self.getpt3().gety())
        else:
            return abs(self.getpt1().gety() - self.getpt2().gety())

    def getsumoftwosides(self):
        """ Return the sum of the two parallel sides of the
            trapezoid """
        if isclose(self.getpt1().gety(), self.getpt2().gety()):
            return abs(self.getpt1().getx() - self.getpt2().getx()) +\
                   abs(self.getpt3().getx() - self.getpt4().getx())
        elif isclose(self.getpt1().gety(), self.getpt3().gety()):
            return abs(self.getpt1().getx() - self.getpt3().getx()) +\
                   abs(self.getpt2().getx() - self.getpt4().getx())
        else:
            return abs(self.getpt1().getx() - self.getpt4().getx()) +\
                   abs(self.getpt2().getx() - self.getpt3().getx())

    def getarea(self):
        """ Return the area of the trapezoid """
        return 0.5*self.getsumoftwosides()*self.getheight()

    def __str__(self):
        """ Return a string representation of the trapezoid """
        return 'Coordinates of the Trapezoid are:\n'\
```

```
                +self.getcoorstr()+'\n'\
                +'Height is: {:.1f}\n'.format(self.getheight())\
                +'Area is: {:.2f}'.format(self.getarea())

class Parallelogram(Trapezoid):
    """ A class representing parallelograms """

    def getwidth(self):
        """ Return the width of the parallelogram """
        if isclose(self.getpt1().gety(), self.getpt2().gety()):
            return abs(self.getpt1().getx() - self.getpt2().getx())
        elif isclose(self.getpt1().gety(), self.getpt3().gety()):
            return abs(self.getpt1().getx() - self.getpt3().getx())
        else:
            return abs(self.getpt1().getx() - self.getpt4().getx())

    def __str__(self):
        """ Return a string representation of the parallelogram """
        return 'Coordinates of the Trapezoid are:\n'\
               +self.getcoorstr()+'\n'\
               +'Width is: {:.1f}\n'.format(self.getwidth())\
               +'Height is: {:.1f}\n'.format(self.getheight())\
               +'Area is: {:.2f}'.format(self.getarea())

class Square(Parallelogram):
    """ A class representing squares """

    def __str__(self):
        """ Return a string representation of the square """
        return 'Coordinates of the Square are:\n'\
               +self.getcoorstr()+'\n'\
               +'Side is: {:.1f}\n'.format(self.getwidth())\
               +'Area is: {:.2f}'.format(self.getarea())
```

(The output is shown on the next page.)

OUTPUT:

```
>>> a = Quadrilateral(3.9, 6.6, 1.4, 4.8, 4.5, 3.8, 2.0, 7.2)
>>> print(a)
Coordinates of the Quadrilateral are:
(3.9, 6.6), (1.4, 4.8), (4.5, 3.8), (2.0, 7.2)
>>> b = Trapezoid(9.5, 0.5, 3.9, 0.5, 4.5, 6.3, 7.3, 6.3)
>>> b.getsumoftwosides()
8.399999999999999
>>> print(b)
Coordinates of the Trapezoid are:
(9.5, 0.5), (3.9, 0.5), (4.5, 6.3), (7.3, 6.3)
Height = 5.8
Area = 24.36
>>> c = Parallelogram(6.0, 3.0, 2.5, 3.0, 9.5, 16.6, 13.0, 16.6)
>>> print(c)
Coordinates of the Trapezoid are:
(6.0, 3.0), (2.5, 3.0), (9.5, 16.6), (13.0, 16.6)
Width = 3.5
Height = 13.6
Area = 47.60
>>> d = Square(5.0, 3.0, 14.5, 3.0, 14.5, 12.5, 5.0, 12.5)
>>> print(d)
Coordinates of the Square are:
(5.0, 3.0), (14.5, 3.0), (14.5, 12.5), (5.0, 12.5)
Side = 9.5
Area = 90.25
```