1. Write a Python program to create a class representing a linked list data structure. Include methods for displaying linked list data, inserting and deleting nodes.

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

    def insertbeg(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def insertend(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

    def deletenode(self, key):
        current = self.head


        if current and current.data == key:
            self.head = current.next
            current = None
            return


        prev = None
        while current and current.data != key:
            prev = current
            current = current.next


        if current is None:
            return

        prev.next = current.next
        current = None

if __name__ == "__main__":
    ll = LinkedList()
    ll.insertend(1)
    ll.insertend(2)
    ll.insertend(3)
    ll.display()

    ll.insertbeg(0)
    ll.display()

    ll.deletenode(2)
    ll.display()
```

2. Write a Python program to create a class representing a queue data structure. Include methods for enqueueing and dequeuing elements.

```python
class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)

    def display(self):
        print(self.items)
q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
q.display()
q.dequeue()
q.display()
```

3. Write a Python program to create a class representing a bank. Include methods for managing customer accounts and transactions.

```python
class Bank:
    def __init__(self):
        self.accounts = {}

    def add_account(self, acc_number, initial_balance=0):
        self.accounts[acc_number] = initial_balance

    def deposit(self, acc_number, amount):
        if acc_number in self.accounts:
            self.accounts[acc_number] += amount

    def withdraw(self, acc_number, amount):
        if acc_number in self.accounts and self.accounts[acc_number] >= amount:
            self.accounts[acc_number] -= amount

    def display_accounts(self):
        for acc_number, balance in self.accounts.items():
            print(f"Account {acc_number}: Balance {balance}")


bank = Bank()
bank.add_account(101, 500)
bank.deposit(101, 200)
bank.withdraw(101, 100)
bank.display_accounts()
```

Create a class "Employee" with attributes name and salary. Implement overloaded operators + and - to combine and compare employees based on their salaries.

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def __add__(self, other):
        return self.salary + other.salary

    def __sub__(self, other):
        return self.salary - other.salary

    def __lt__(self, other):
        return self.salary < other.salary

    def __gt__(self, other):
        return self.salary > other.salary
emp1 = Employee("Alice", 5000)
emp2 = Employee("Bob", 6000)
print(emp1 + emp2)
print(emp1 - emp2)
print(emp1 > emp2)
```

5. Create a base class "Shape" with methods to calculate the area and perimeter. Implement derived classes "Rectangle" and "Circle" that inherit from "Shape" and provide their own area and perimeter calculations.

```python
class Shape:
    def area(self):
        pass

    def perimeter(self):
        pass

class Rectangle(Shape):
    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)

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14 * self.radius

rect = Rectangle(4, 5)
circle = Circle(3)
print(rect.area())
print(circle.perimeter())
```

6. Create a class "BankAccount" with attributes account number and balance. Implement methods to deposit and withdraw funds, and a display method to show the account details.

```python
class BankAccount:
    def __init__(self, acc_num, balance=0):
        self.acc_num = acc_num
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount

    def display(self):
        print(f"Account {self.acc_num}: Balance {self.balance}")

acc = BankAccount(12345, 1000)
acc.deposit(500)
acc.withdraw(200)
acc.display()
```

7. Create a class for representing any 2-D point or vector. The methods inside this class include its magnitude and its rotation with respect to the X-axis. Using the objects define functions for calculating the distance between two vectors, dot product, cross product of two vectors. Extend the 2-D vectors into 3-D using the concept of inheritance. Update the methods according to 3-D.

```python
import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)

    def rotate(self, angle):
        rad = math.radians(angle)
        cos_a, sin_a = math.cos(rad), math.sin(rad)
        return Vector2D(self.x * cos_a - self.y * sin_a, self.x * sin_a + self.y * cos_a)

    def distance(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

    def dot(self, other):
        return self.x * other.x + self.y * other.y

    def cross(self, other):
        return self.x * other.y - self.y * other.x

class Vector3D(Vector2D):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.z = z

    def magnitude(self):
        return math.sqrt(self.x**2 + self.y**2 + self.z**2)

v2d_1 = Vector2D(1, 2)
v2d_2 = Vector2D(3, 4)
print(v2d_1.magnitude())
print(v2d_1.distance(v2d_2))

v3d = Vector3D(1, 2, 3)
print(v3d.magnitude())
```

8. Decode the message:
   A message containing the letters from A-Z can be encoded into the numbers using the mapping A-> 1, B-> 2, C-> 3, ..., Z-> 26. To decode an encoded message, you need to group the digits and do the reverse mapping. You are required to display all the possible decoded messages. For example: "11106" can be decoded into:
   a. "AAJF" with the grouping (1 1 10 6)
   b. "KJF" with the grouping (11 10 6)

```python
def decode_message(encoded_message):
    def _decode(index, current_decoding):
        if index == len(encoded_message):
            result.append(current_decoding)
            return

        if encoded_message[index] == '0': #handle leading 0s
          return

        # Try decoding one digit
        digit = int(encoded_message[index])
        if 1 <= digit <= 26:
            _decode(index + 1, current_decoding + chr(digit + 64))  # 64 is ASCII for 'A'

        # Try decoding two digits
        if index + 1 < len(encoded_message):
            two_digits = int(encoded_message[index:index+2])
            if 10 <= two_digits <= 26:
                _decode(index + 2, current_decoding + chr(two_digits + 64))

    result = []
    _decode(0, "")
    return result

encoded_message = "11106"
decoded_messages = decode_message(encoded_message)
print(decoded_messages)

encoded_message = "226"
decoded_messages = decode_message(encoded_message)
print(decoded_messages)

encoded_message = "06"
decoded_messages = decode_message(encoded_message)
print(decoded_messages)
encoded_message = "10"
decoded_messages = decode_message(encoded_message)
print(decoded_messages)
```

9. Create a tokenizer for your own language (mother tongue you speak). The tokenizer should tokenize punctuations, dates, urls, emails, numbers (in all different forms such as “33.15”,

```python
import re

def tokenize(text):
    tokens = []
    patterns = [
        r"\d{1,2}/\d{1,2}/\d{2,4}",  # Dates
        r"https?://\S+|www\.\S+",    # URLs
        r"\b\w+@\w+\.\w+\b",         # Emails
        r"[\w\.-]+",                 # Words, numbers, social media handles
        r"\.\d+|\d+\.?\d*",          # Numbers
        r"[,!?;]"                    # Punctuation
    ]

    for pattern in patterns:
        matches = re.findall(pattern, text)
        for match in matches:
            tokens.append(match)
            text = text.replace(match, '')
    return tokens

text = "Email: test@example.com, URL: https://example.com, Date: 23/05/2020"
print(tokenize(text))
```