No.1

```
class Clock:
    def __init__(self, hour, minute, second):
        self.hour = hour
        self.minute = minute
        self.second = second

    def set_hour(self, hour):
        self.hour = hour

    def set_minute(self, minute):
        self.minute = minute


    def set_seconds(self, second):
        self.second = second

    def set_time(self, hour, minute, second):
        self.set_hour(hour)
        self.set_minute(minute)
        self.set_seconds(second)

    def tick(self):
        self._second += 1
        if (self._second == 60):
            self._second = 0
            self._minute += 1

            if (self._minute == 60):

                self._minute = 0
                self._hour += 1

                if (self._hour == 24):
                    self._hour = 0

    def get_time(self):
        if(self.hour > 0 and self.hour <= 12):

            state = "AM"
        elif(self.hour > 12 and self.hour < 24):
            self.hour -= 12
            state = "PM"

        elif(self.hour == 12):
            self.hour = 0
            state = "PM"

        elif(self.hour == 24):
            self.hour = 0
            state = "AM"
```

```
        print(f"{self.hour:02d}:{self.minute:02d}:{self.second:02d} {state}")


def main():
    clock = Clock(12, 50, 12)
    clock.set_time(9,30,24)
    clock.set_hour(13)
    clock.set_seconds(30)
    clock.get_time()

main()
```

No.2

```
class Poly:
    def __init__(self, x):
        self.x = list(x)

    def who_is_bigger(self, p):
        first_power = self.find_powerOfx()
        first_coef = self.x

        second_power = p.find_powerOfx()
        second_coef = p.x



        if (len(first_power) >= len(second_power)):

            bigger_coef = first_coef
            bigger_power = first_power

            smaller_coef = second_coef
            smaller_power = second_power

        elif (len(first_power) < len(second_power)):

            bigger_coef = second_coef
            bigger_power = second_power

            smaller_coef = first_coef
            smaller_power = first_power


        while(len(smaller_power) != len(bigger_power)):
            smaller_power += (0,)
            smaller_coef += (0,)
```

```
        return (smaller_coef, smaller_power, bigger_coef, bigger_power)


    def add(self, p):
        (smaller_coef, smaller_power, bigger_coef, bigger_power) = self.who_is_bigger(p)

        new_coef = []
        for i in range(0, len(bigger_power)):

            if bigger_power[i] == smaller_power[i]:
                new_coef += (smaller_coef[i] + bigger_coef[i],)

            else:
                new_coef += (bigger_coef[i],)

        self.x = new_coef
        return Poly(self.x)

    def scalar_mulltiply(self, n):

        for i in range(len(self.x)):
            self.x[i] *= n

        return Poly(self.x)

    def multiply(self, p):
        #(smaller_coef, smaller_power, bigger_coef, bigger_power) = self.who_is_bigger(p)

        new_coef = []

        # print(len(self.x) + len(p.x) - 1)
        for i in range(0, (len(self.x) + len(p.x)) -1 ):
            new_coef += (0, )
        # [0 + 0] += 1*1
        # [0 + 1] += 1*1
        # [1 + 0] += 1*1
        # [1 + 1] += 1*1
        # [2 + 0] += 1*1
        # [2 + 1] += 1*1
        # print(f"new_coef = {new_coef}")

        i = 0
        while(i != len(self.x)):
            j = 0
            while(j != len(p.x)):
                new_coef[i+j] += (self.x[i] * p.x[j])
                j += 1
            i += 1

        # print(f"new_coef = {new_coef}")


        return Poly(new_coef)
```

```
def power(self, n):

    new_coef = []

    # print(len(self.x) + len(self.x))
    for i in range(0, (len(self.x) *n ) -1 ):
        new_coef += (0, )
    # [0 + 0] += 1*1
    # [0 + 1] += 1*1
    # [1 + 0] += 1*1
    # [1 + 1] += 1*1
    # [2 + 0] += 1*1
    # [2 + 1] += 1*1
    # print(f"new_coef = {new_coef}")

    i = 0
    while(i != len(self.x)):
        j = 0
        while(j != len(self.x)):
            new_coef[i+j] += (self.x[i] * self.x[j])
            j += 1
        i += 1

    # print(f"new_coef = {new_coef}")

    return Poly(new_coef)

def find_powerOfx(self):
    count = 0
    powers = ()
    for i in self.x:

        if (i == 0):
            powers += (0,)
            count += 1
            continue
        else:
            powers += (count,)
            count += 1

    return powers


def diff(self):

    new_coef = ()

    for i in range(len(self.x)-1):

        new_coef += (self.x[i + 1] * (i + 1),)
```

```
        return Poly(new_coef)

    def integrate(self):
        # ∫ x^n dx = (( x^(n+1) ) / (n+1))+C ; n≠1

        new_coef = []
        # print(f"len(self.x) = {len(self.x)}")
        for i in range(0, len(self.x)+1):
            new_coef += (0, )


        # print(f"new_coef = {new_coef}")
        original_powers = self.find_powerOfx()


        print()


        for i in range(0, len(self.x)):
            # print(f"self.x[i] / original_powers[i] + 1  = {self.x[i] // original_powers[i] +
1 }")

            new_coef[i+1] = (self.x[i] / (original_powers[i] + 1 ))



            # print(f"self.x[i] : {self.x[i]}")
            # print(f"original_powers[i] : {original_powers[i] + 1}")
            # print()

        # print(f"new_coef = {new_coef}")

        return Poly(new_coef)

    def print(self):
        count = 0

        for i in self.x:
            sign = "+"
            if i < 0:
                sign = "-"
            if ( count == 0):
                sign = ""
            if (i == 0):
                count += 1
                continue
            else:
                i = abs(i)
                if count == 0:
                    print(f"{sign}{int(i)} ", end = "")
                else:
                    print(f"{sign} {int(i)}x^{count} ", end = " ")
                count += 1
```

```
        print()

    def eval(self, n):
        count = 0
        total = 0
        for i in self.x:

            if (i == 0):

                # print(f"Skipped {count}^th power since it's coeff is 0.")
                count += 1
                continue
            else:
                # print(f"total = {total}")
                total += i * (n ** count)
                # print(f"{i} * ({n} ** {count}) = {total}")
                count += 1

        print(total)


def main():
    #          0th, 1th, 2th, 3th, 4th, ....
    p = Poly( (  1, 2, 3) )
    p.print()

    print()
    p.diff().print()
    p.integrate().print()


    # q = p.power(2)
    # q.print()
    # p.print()

    # p.eval(3)
    # print(p.find_powerOfx())
    # print(q.find_powerOfx())

    # r = p.add(q)
    # r.print()

    # p.scalar_mulltiply(2)
    # p.print()

    # s = Poly((1,1))
    # s.print()
    # r = p.multiply(s)
    # r.print()

    # p.diff()
    # print(f"Diff of p : ", end = "")
    # p.print()
```

```
    # r = p.power(2)
    # r.print()




main()
```

No.3

```
class LinearEquation:
    def __init__(self, a, b, c, d, e, f):
        self.__a = a
        self.__b = b
        self.__c = c
        self.__d = d
        self.__e = e
        self.__f = f

    def get_a(self, a):
        return self.__a

    def get_b(self, b):
        return self.__b

    def get_c(self, c):
        return self.__c

    def get_d(self, d):
        return self.__d

    def get_e(self, e):
        return self.__e

    def get_f(self, f):
        return self.__f

    def isSolvable(self):
        if( (self.__a * self.__d) - (self.__b* self.__c) !=0 ):
            return True
        else:
            return False

    def getX(self):
        top =( self.__e * self.__d) - (self.__b * self.__f)
        bottom = (self.__a * self.__d) - (self.__b * self.__c)
        return top / bottom

    def getY(self):
        top =( self.__a * self.__f) - (self.__e * self.__c)
        bottom = (self.__ * self.__d) - (self.__b * self.__c)
        return top / bottom
```