



Homework # 7

**01286121 Computer Programming
Software Engineering Program,
Department of Computer Engineering,
School of Engineering, KMITL**

By

66011203 Sai Marn Pha

1. Define a Clock class in Python, whose properties are hour, minute, second; and it provides methods to set time, get time, tick (increment the current time by 1 second), and display time in am/pm format.

The source code for problem 1:

```
class Clock:

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

        self.set_time(hour, minute, second)

    def set_time(self, hr , mn, sec) :

        if hr >= 24 or mn > 60 or sec > 60:
            exit("Invalid time format")

        self.hour = hr
        if hr == 24 :
            self.hour = 0
        if mn == 60 :
            self.minute = 0
            if self.hour < 23 :
                self.hour = hr+1
            else:
                self.hour = 0

        else :
            self.minute = mn
        if sec >= 60 :
            self.second = sec - self.second
        else :
            self.second = sec

    def get_time (self) :

        hr = self.hour
        mn = self.minute
        sec = self.second
        pm_am = 'am'
        if self.hour < 12 and self.hour > 0: #1-am to 11 am
            hr = self.hour
            pm_am = 'am'
        elif self.hour == 12 : #12 pm
```

```

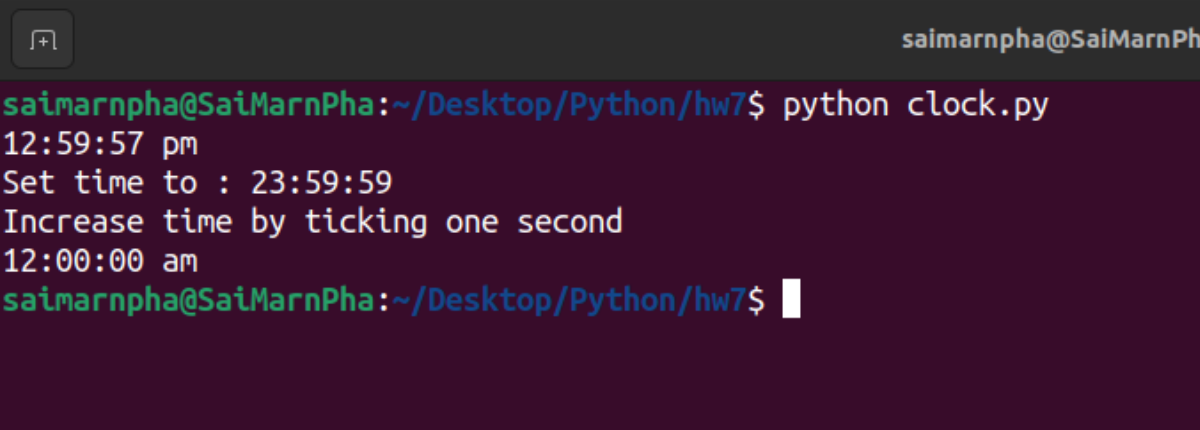
        hr = self.hour
        pm_am = "pm"
    elif self.hour > 12 and self.hour < 24: #13 - 23
        hr = self.hour - 12
        pm_am = "pm"
    elif self.hour == 0 or self.hour == 24: # midnight
        hr = 12
        pm_am = 'am'
    return f'{hr:02}:{mn:02}:{sec:02} {pm_am}'

def tick(self) :
    self.second += 1
    if self.second == 60 :
        self.second = 0
        self.set_time(self.hour, self.minute +1, self.second)

myclock = Clock(12,59,59)
myclock.set_time(0,59,59)
myclock.tick()
myclock.tick()
print(myclock.get_time())

```

The result of running the source code :



A terminal window with a dark purple background. The prompt is 'saimarnpha@SaiMarnPh'. The user enters 'python c\lock.py'. The output shows the current time '12:59:57 pm', then 'Set time to : 23:59:59', followed by 'Increase time by ticking one second', and finally '12:00:00 am'. The prompt returns to 'saimarnpha@SaiMarnPh:~/Desktop/Python/hw7\$'.

```

saimarnpha@SaiMarnPh:~/Desktop/Python/hw7$ python c\lock.py
12:59:57 pm
Set time to : 23:59:59
Increase time by ticking one second
12:00:00 am
saimarnpha@SaiMarnPh:~/Desktop/Python/hw7$

```

2. A single-variable polynomial can be represented in Python as a tuple of coefficients. For example, the polynomial $14 + 7x - 5x^2 + 18x^3$ can be represented as (14, 7, -5, 0, 0, 18).

Define the class Poly which stores a polynomial in variable x represented in tuple format and provides the following methods:

Methods:

- `add(p)`: given a Poly object `p`, returns the result of adding itself with `p`
- `scalar_multiply(n)`: given a number `n`, returns the result of multiply `c` with itself
- `multiply(p)`: given a Poly object `p`, returns the result of multiply itself with `p`
- `power(n)`: given a natural number $n \geq 0$, return the Poly object resulted from taking the `n`th-power
- `diff()`: returns the Poly object resulted from differentiating the stored polynomial with respect to
- `integrate()`: returns the Poly object resulted from integrating the stored polynomial with respect to x (the constants resulted from the integration can be assumed to be 0)
- `eval(n)`: given a number `n`, evaluate the polynomial with $x = n$
- `print()`: print out the stored polynomial in a pretty format (e.g. $14 + 7x - 5x^2 + 18x^5$)

The class may contain additional properties and methods not described here.

The construction of an object of class Poly should accept a tuple representing a polynomial and store it in, the object.

For example:

```
>>> p = Poly ( (1, 0, -2))
>>> p.print ()
1 - 2x^2
>>> g = p. power (2)
>>> g.print ( )
1 - 4x^2 + 4x^4
>>> p.eval (3)
-17
>>> r = p. add (g)
>>> r.print()
2 - 6x^2 + 4x^4
>>> r.diff () . print ()
-12x + 16x^3
```

The source code for the problem 2 :

```
class Poly :

    def __init__(self, polynomial : tuple = ()):

        if type(polynomial) != tuple :
            exit("Polynomial argument must be in tuple")
        self.x = polynomial

    def print(self) :

        y = ""
        for (i, elem) in enumerate(self.x) :
            if elem == 0 :
                continue
            if i == 0 :
                y += f'{elem}'
            elif i == 1:
                elem = "{:}".format(elem)
                y += f'{elem}x'
            else :
                elem = "{:}".format(elem)
                y += f'{elem}x^{i}'
        if y[0] == '+' :
            y=y[1:len(y)]
        print(y)

    def scalar_multiply(self, n) :
        y = []
        for i, elem in enumerate(self.x) :
            y.append(elem * n)
        self.x = tuple(y)
        return self

    def multiply(self, P) :
        if type(P) != Poly :
            exit("P argument must be instance of Poly class")

        P = list(P.x)
        x = list(self.x)
        y = [0] * (len(P) * len(x))

        for (a, elem) in enumerate(P) :
            for (b, elem2) in enumerate(x) :
                y[ a+b ] += elem * elem2
```

```

for i in range(len(y)-1, -1, -1) :
    if y[i] == 0 :
        y.pop(i)
    else :
        break
return tuple(y)

```

```

def power(self, n) :

```

```

    mulitpler = self
    for i in range(1, n) :
        multiply_res = self.multiply(mulitpler)
        mulitpler = Poly(multiply_res)
    return mulitpler

```

```

def eval(self, n) :

```

```

    summation = 0
    for i, elem in enumerate(self.x) :
        summation += elem * n**i
    return summation

```

```

def diff(self) :

```

```

    y = list(self.x)
    res = [0] * (len(y)-1)
    for i in range(0, len(y)) :
        if i == 0 :
            res[i]=0
        else :
            res[i-1]= y[i] * i
    self.x = tuple(res)
    return self

```

```

def add(self, p) :

```

```

    larger_poly = self.x
    smaller_poly = p.x

    if len(p.x) >= len(self.x) :
        larger_poly = p.x
        smaller_poly = self.x

```

```

    larger_poly = list(larger_poly)
    smaller_poly = list(smaller_poly)

```

```

    for i in range(0, len(smaller_poly)) :
        larger_poly[i] += smaller_poly[i]

```

```
self.x = tuple(larger_poly)
return self
```

```
def integrate(self) :
```

```
    poly = list(self.x)
    poly.append(0)
    res = [0] * len(poly)
    for i in range(0, len(poly)-1) :
        index = i+1
        res[index] = round(poly[i]/index, 2)
```

```
self.x = tuple(res)
return self
```

```
pol = Poly((1,0,-2))
pol.print()
q = pol.power(2)
q.print()
```

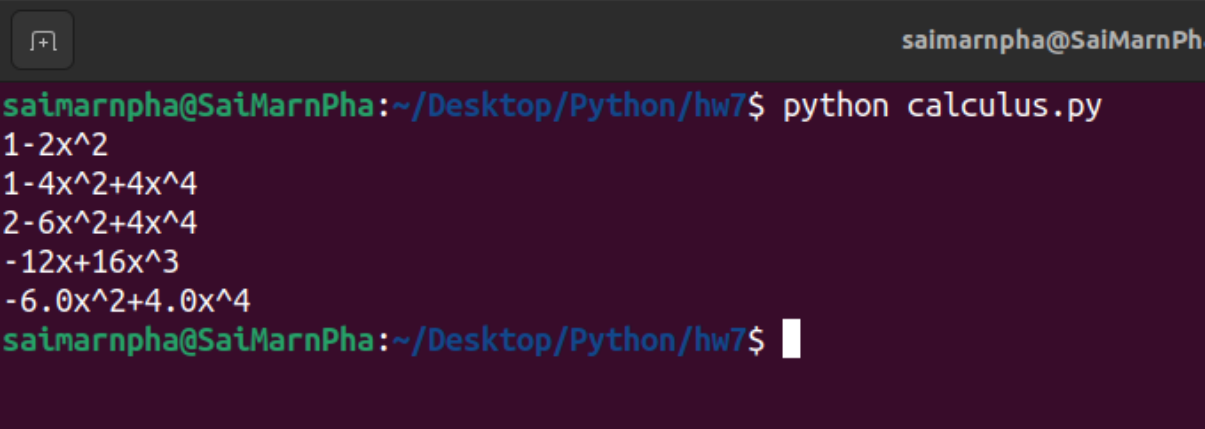
```
pol.eval(3)
```

```
r = pol.add(q)
r.print()
```

```
r.diff().print()
```

```
pol.integrate().print()
```

The result of running the source code :



```
saimarnpha@SaiMarnPha:~/Desktop/Python/hw7$ python calculus.py
1-2x^2
1-4x^2+4x^4
2-6x^2+4x^4
-12x+16x^3
-6.0x^2+4.0x^4
saimarnpha@SaiMarnPha:~/Desktop/Python/hw7$
```

3. Design a class named LinearEquation for a 2 X 2 system of linear equations:

$$ax + by = e \quad x = (ed - bf) / (ad - bc)$$

$$cx + dy = f \quad y = (af - ec) / (ad - bc)$$

The class contains:

- The private data fields a, b, c, d, e, and f with get methods.
- A constructor with the arguments for a, b, c, d, e, and f.
- Six get methods for a, b, c, d, e, and f.
- A method named isSolvable() that returns true if ad - bc is not 0.
- The methods getX() and getY() that return the solution for the equation.

The source code for the problem 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) :
```

```
    return self.__a
```

```
def get_b(self) :
```

```
    return self.__b
```

```
def get_c(self) :
```

```
    return self.__c
```

```
def get_d(self) :
```

```
    return self.__d
```

```
def get_e(self) :
```

```
    return self.__e
```

```
def get_f(self) :
```

```
    return self.__f
```

```
def isSolvable(self) :
```

```
    return not (( self.__a * self.__d - self.__b * self.__c ) == 0 )
```

```
def getX(self) :
```

```
    x = (self.__e * self.__d - self.__b * self.__f) / (self.__a * self.__d - self.__b * self.__c)
```

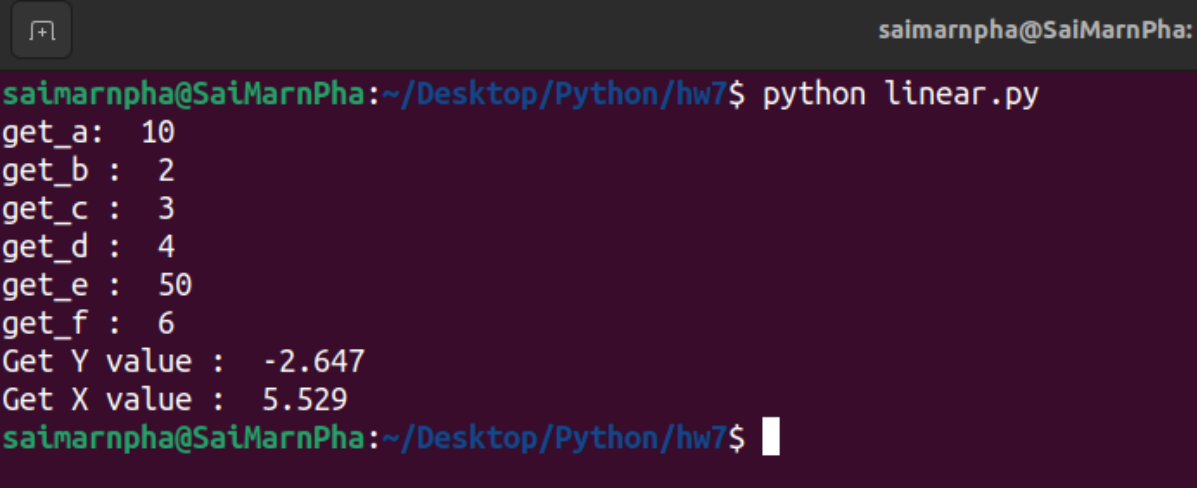
```
    return round(x, 3)
```



```
def getY(self) :  
    y = (self.__a * self.__f - self.__e * self.__c) / (self.__a * self.__d - self.__b * self.__c)  
    return round(y, 3)
```

```
lq = LinearEquation(10,2,3,4,50,6)  
print("get_a: ", lq.get_a())  
print("get_b : ", lq.get_b())  
print("get_c : ", lq.get_c())  
print("get_d : ", lq.get_d())  
print("get_e : ", lq.get_e())  
print("get_f : ", lq.get_f())  
print("Get Y value : ",lq.getY())  
print("Get X value : ",lq.getX())
```

The result of running the source code :



```
saimarnpha@SaiMarnPha:~/Desktop/Python/hw7$ python linear.py  
get_a: 10  
get_b : 2  
get_c : 3  
get_d : 4  
get_e : 50  
get_f : 6  
Get Y value : -2.647  
Get X value : 5.529  
saimarnpha@SaiMarnPha:~/Desktop/Python/hw7$
```