National Sun Yat-sen University
Introduction To Blockchain Technology
Homework 2
Course Number: CSE222, Chapter: 3 & 4

# Notice :
1. No late homework.
2. Please submit your homework to **Cyber University of National Sun Yat-sen University** (https://cu.nsysu.edu.tw/mooc/index.php). It is not allowed to submit assignments to any other location.
3. You only need to submit **Homework 2.docx**, after you paste all screenshot of your code and code execution results of solving following problems.
4. We only accept using **python** to write program files.
5. **Please answer each question according to the requirements below, otherwise no points will be awarded.**

1. Verify whether the signature is valid:

P=(0x801be5a7c4faf73dd1c3f28cebf78d6ba7885ead88879b76ffb815d59056af14,0x826ddfcc38dafe6b8d463b609facc009083c8173e21c5fc45b3424964e85f49e)

Signature:

z=0x90d7aecf3f2855d60026f10faab852562c76e7e043cf243474ba5018447c2c22

r=0xf01d6b9018ab421dd410404cb869072065522bf85734008f105cf385a023a80f

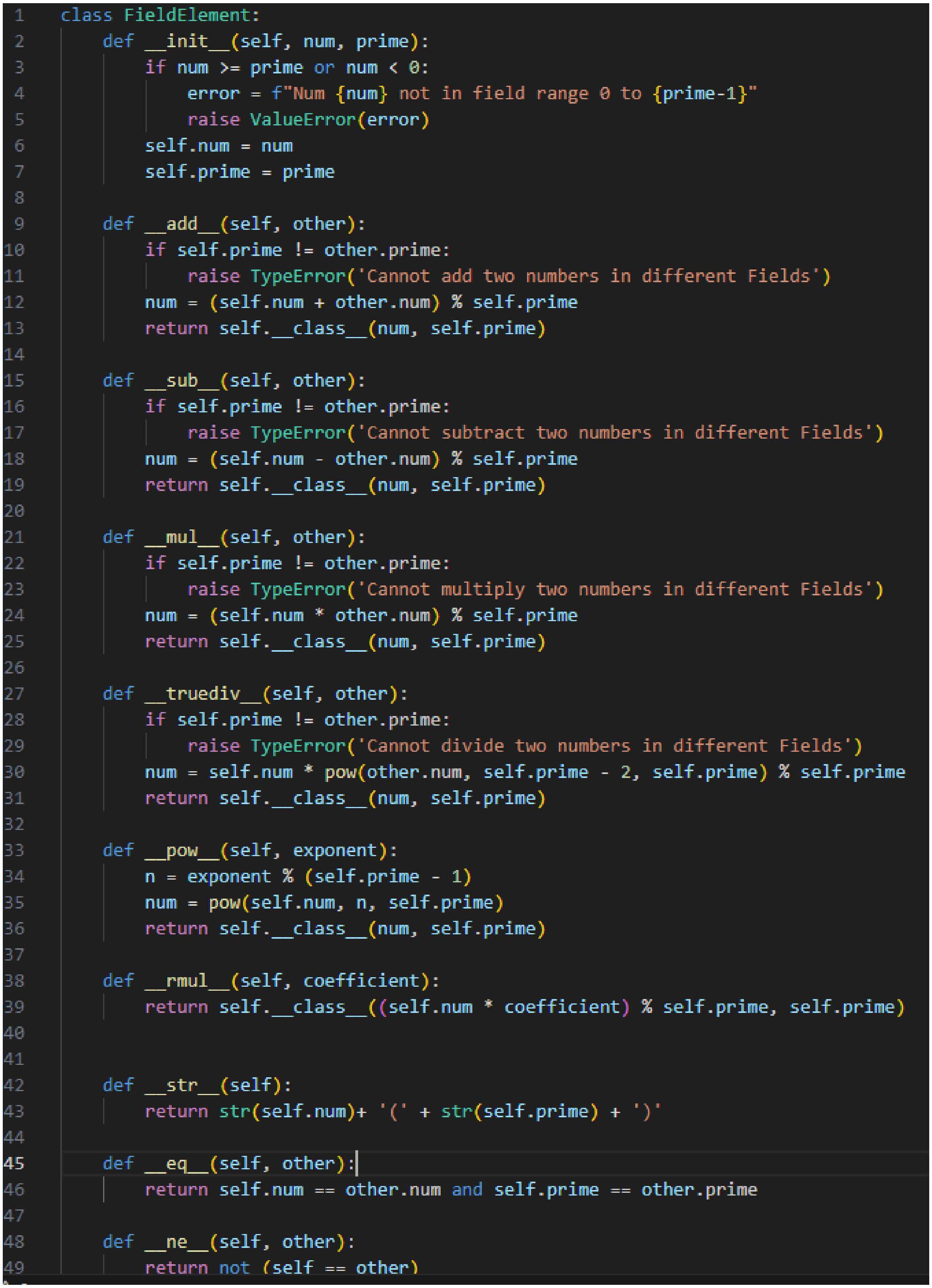
s=0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8

# Grading for Problem 1:
- Total: 20%
  - Correct code and code execution result: 20%
    (Only have correct code and correct code execution can get score.)

--- Please paste screenshot of your code and code execution result of solving problem 1 below. ---

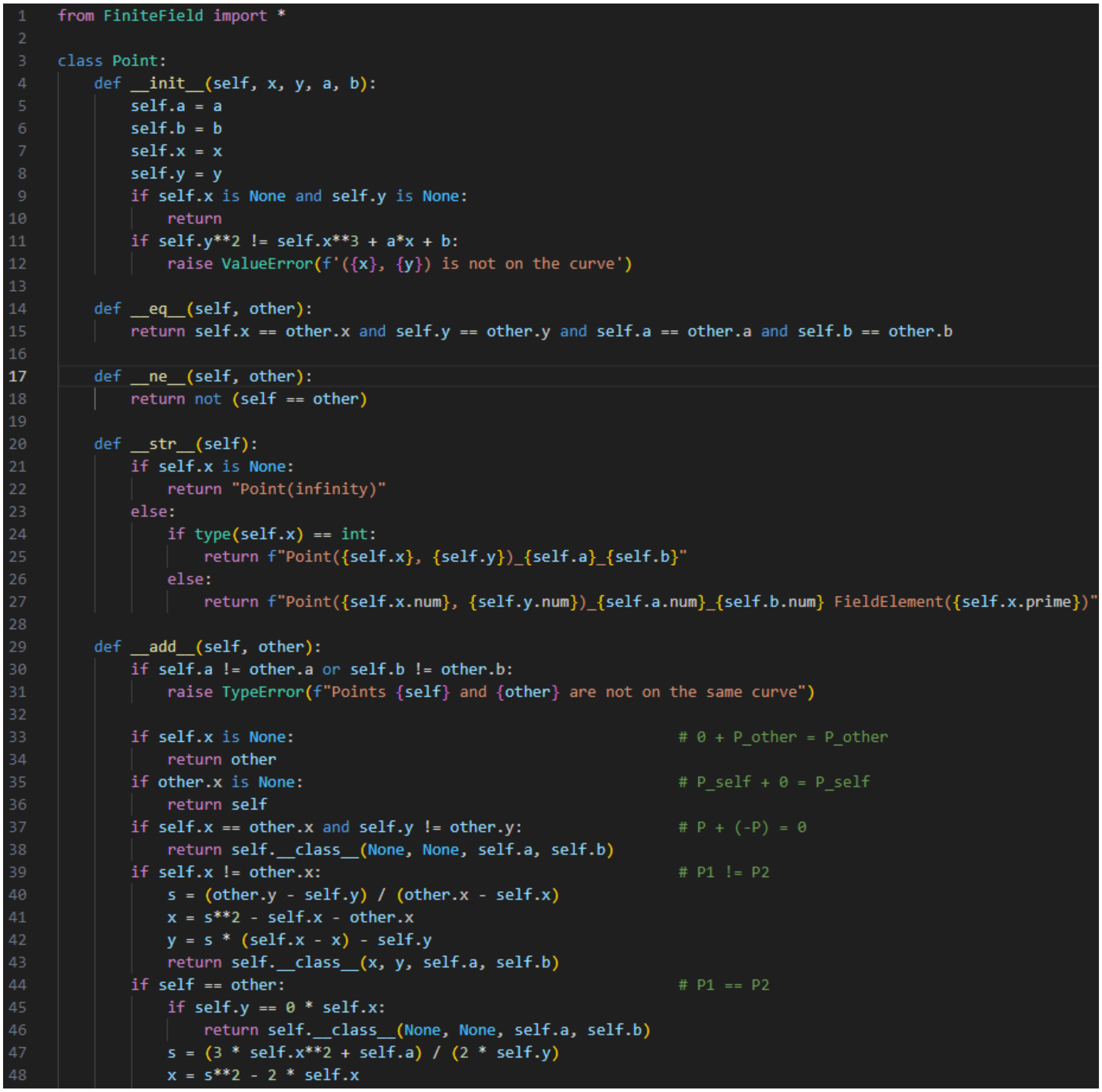
Code:

FiniteField.py(包含 class FieldElement):

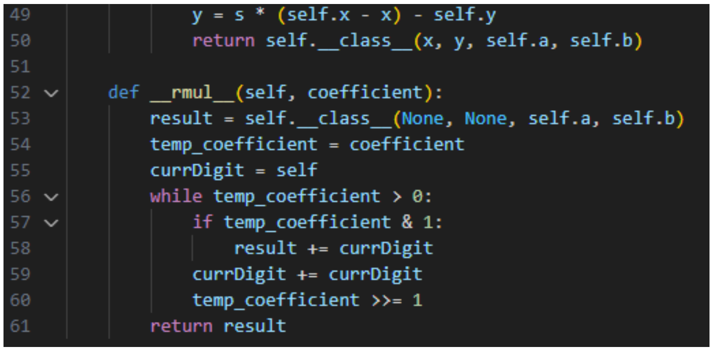


```python
class FieldElement:
    def __init__(self, num, prime):
        if num >= prime or num < 0:
            error = f"Num {num} not in field range 0 to {prime-1}"
            raise ValueError(error)
        self.num = num
        self.prime = prime

    def __add__(self, other):
        if self.prime != other.prime:
            raise TypeError('Cannot add two numbers in different Fields')
        num = (self.num + other.num) % self.prime
        return self.__class__(num, self.prime)

    def __sub__(self, other):
        if self.prime != other.prime:
            raise TypeError('Cannot subtract two numbers in different Fields')
        num = (self.num - other.num) % self.prime
        return self.__class__(num, self.prime)

    def __mul__(self, other):
        if self.prime != other.prime:
            raise TypeError('Cannot multiply two numbers in different Fields')
        num = (self.num * other.num) % self.prime
        return self.__class__(num, self.prime)

    def __truediv__(self, other):
        if self.prime != other.prime:
            raise TypeError('Cannot divide two numbers in different Fields')
        num = self.num * pow(other.num, self.prime - 2, self.prime) % self.prime
        return self.__class__(num, self.prime)

    def __pow__(self, exponent):
        n = exponent % (self.prime - 1)
        num = pow(self.num, n, self.prime)
        return self.__class__(num, self.prime)

    def __rmul__(self, coefficient):
        return self.__class__((self.num * coefficient) % self.prime, self.prime)


    def __str__(self):
        return str(self.num)+ '(' + str(self.prime) + ')'

    def __eq__(self, other):
        return self.num == other.num and self.prime == other.prime

    def __ne__(self, other):
        return not (self == other)
```

EllipticCurves.py(包含 class Point):

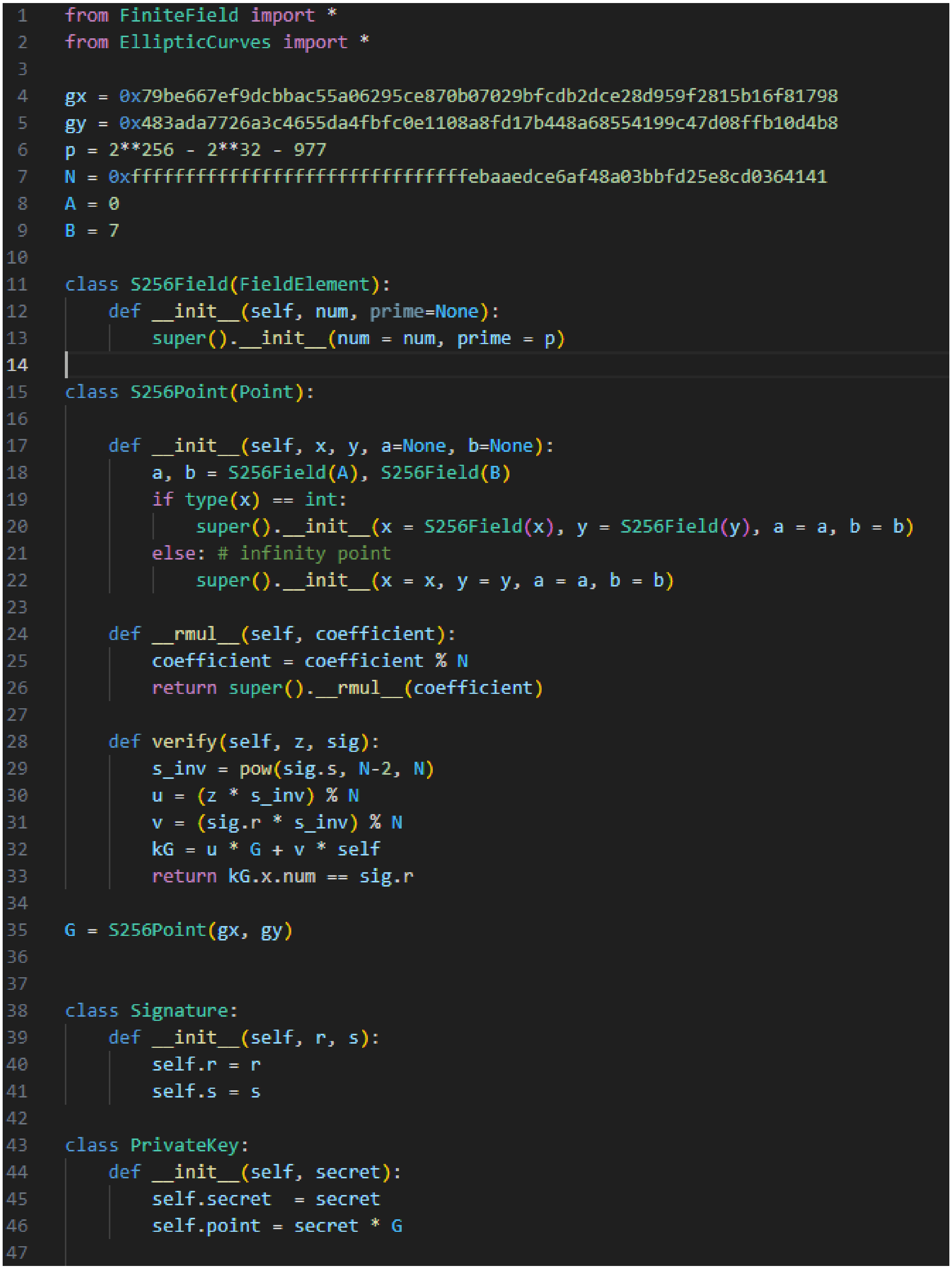


```python
from FiniteField import *

class Point:
    def __init__(self, x, y, a, b):
        self.a = a
        self.b = b
        self.x = x
        self.y = y
        if self.x is None and self.y is None:
            return
        if self.y**2 != self.x**3 + a*x + b:
            raise ValueError(f'({x}, {y}) is not on the curve')

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y and self.a == other.a and self.b == other.b

    def __ne__(self, other):
        return not (self == other)

    def __str__(self):
        if self.x is None:
            return "Point(infinity)"
        else:
            if type(self.x) == int:
                return f"Point({self.x}, {self.y})_{self.a}_{self.b}"
            else:
                return f"Point({self.x.num}, {self.y.num})_{self.a.num}_{self.b.num} FieldElement({self.x.prime})"

    def __add__(self, other):
        if self.a != other.a or self.b != other.b:
            raise TypeError(f"Points {self} and {other} are not on the same curve")

        if self.x is None:                                      # 0 + P_other = P_other
            return other
        if other.x is None:                                     # P_self + 0 = P_self
            return self
        if self.x == other.x and self.y != other.y:             # P + (-P) = 0
            return self.__class__(None, None, self.a, self.b)
        if self.x != other.x:                                   # P1 != P2
            s = (other.y - self.y) / (other.x - self.x)
            x = s**2 - self.x - other.x
            y = s * (self.x - x) - self.y
            return self.__class__(x, y, self.a, self.b)
        if self == other:                                       # P1 == P2
            if self.y == 0 * self.x:
                return self.__class__(None, None, self.a, self.b)
            s = (3 * self.x**2 + self.a) / (2 * self.y)
            x = s**2 - 2 * self.x
```

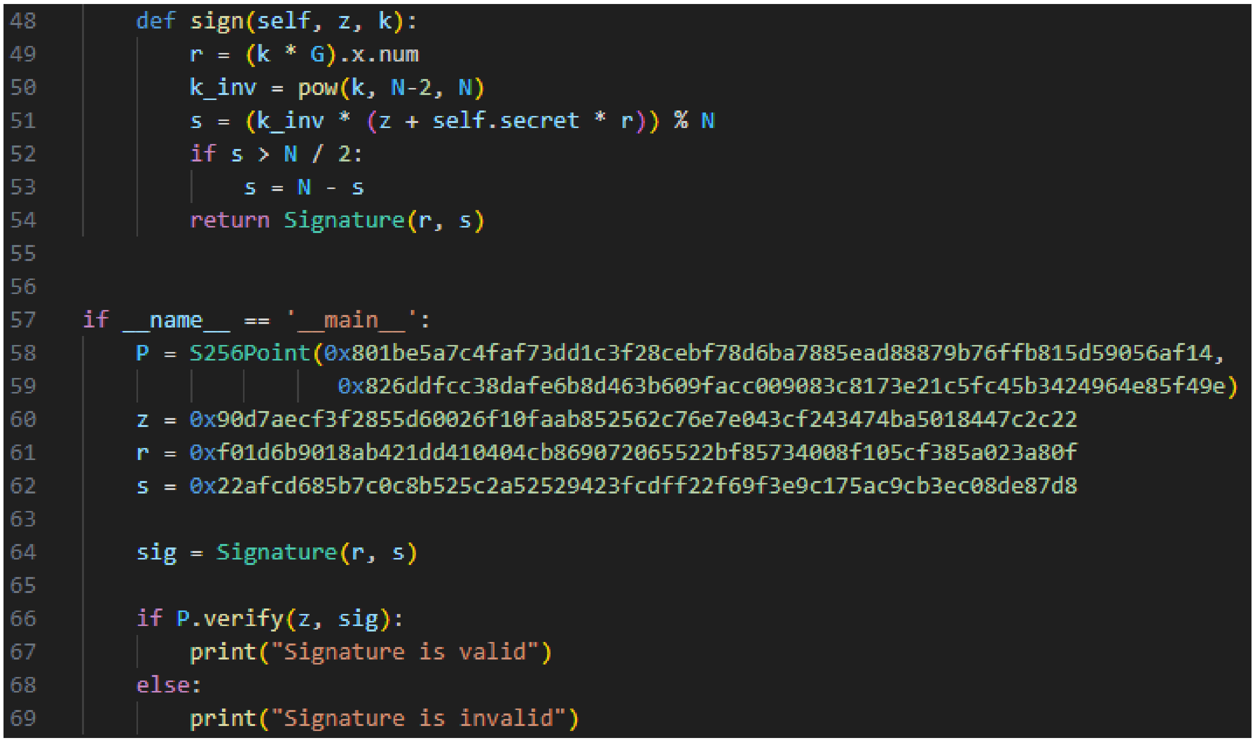


```python
            y = s * (self.x - x) - self.y
            return self.__class__(x, y, self.a, self.b)

    def __rmul__(self, coefficient):
        result = self.__class__(None, None, self.a, self.b)
        temp_coefficient = coefficient
        currDigit = self
        while temp_coefficient > 0:
            if temp_coefficient & 1:
                result += currDigit
            currDigit += currDigit
            temp_coefficient >>= 1
        return result
```

Verify.py(包含 class S256Field, S256Point, Signature, PrivateKey):

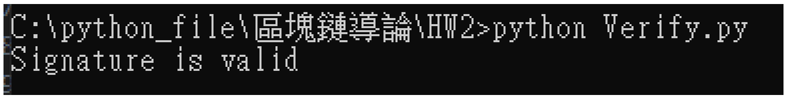


```python
from FiniteField import *
from EllipticCurves import *

gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
p = 2**256 - 2**32 - 977
N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
A = 0
B = 7

class S256Field(FieldElement):
    def __init__(self, num, prime=None):
        super().__init__(num = num, prime = p)

class S256Point(Point):

    def __init__(self, x, y, a=None, b=None):
        a, b = S256Field(A), S256Field(B)
        if type(x) == int:
            super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
        else: # infinity point
            super().__init__(x = x, y = y, a = a, b = b)

    def __rmul__(self, coefficient):
        coefficient = coefficient % N
        return super().__rmul__(coefficient)

    def verify(self, z, sig):
        s_inv = pow(sig.s, N-2, N)
        u = (z * s_inv) % N
        v = (sig.r * s_inv) % N
        kG = u * G + v * self
        return kG.x.num == sig.r

G = S256Point(gx, gy)


class Signature:
    def __init__(self, r, s):
        self.r = r
        self.s = s

class PrivateKey:
    def __init__(self, secret):
        self.secret  = secret
        self.point = secret * G

```



```python
    def sign(self, z, k):
        r = (k * G).x.num
        k_inv = pow(k, N-2, N)
        s = (k_inv * (z + self.secret * r)) % N
        if s > N / 2:
            s = N - s
        return Signature(r, s)


if __name__ == '__main__':
    P = S256Point(0x801be5a7c4faf73dd1c3f28cebf78d6ba7885ead88879b76ffb815d59056af14,
                  0x826ddfcc38dafe6b8d463b609facc009083c8173e21c5fc45b3424964e85f49e)
    z = 0x90d7aecf3f2855d60026f10faab852562c76e7e043cf243474ba5018447c2c22
    r = 0xf01d6b9018ab421dd410404cb869072065522bf85734008f105cf385a023a80f
    s = 0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8

    sig = Signature(r, s)

    if P.verify(z, sig):
        print("Signature is valid")
    else:
        print("Signature is invalid")
```

Result:



```
C:\python_file\區塊鏈導論\HW2>python Verify.py
Signature is valid
```

2. Sign the following message with the secret:

Private key e=1234567

z = int.from_bytes(hash256(b'Introduction to Bitcoin homework 2.2'), 'big')

k = 1234567

Print the point, z, r and s in hexadecimal.

# Grading for Problem 2:

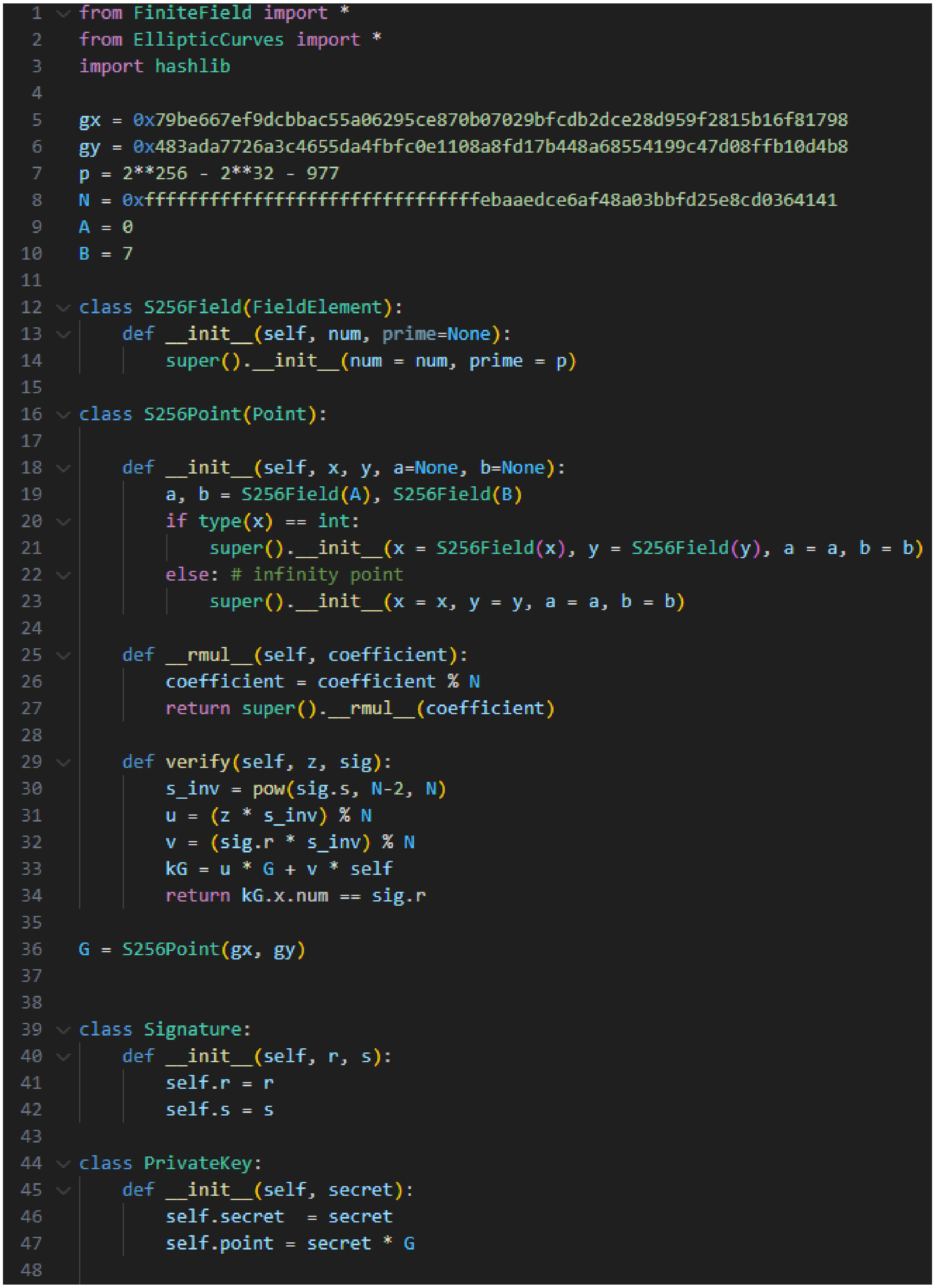
- Total: 20%
  - Correct code and code execution result: 20%
    (Only have correct code and correct code execution can get score.)

--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 2 below. ---

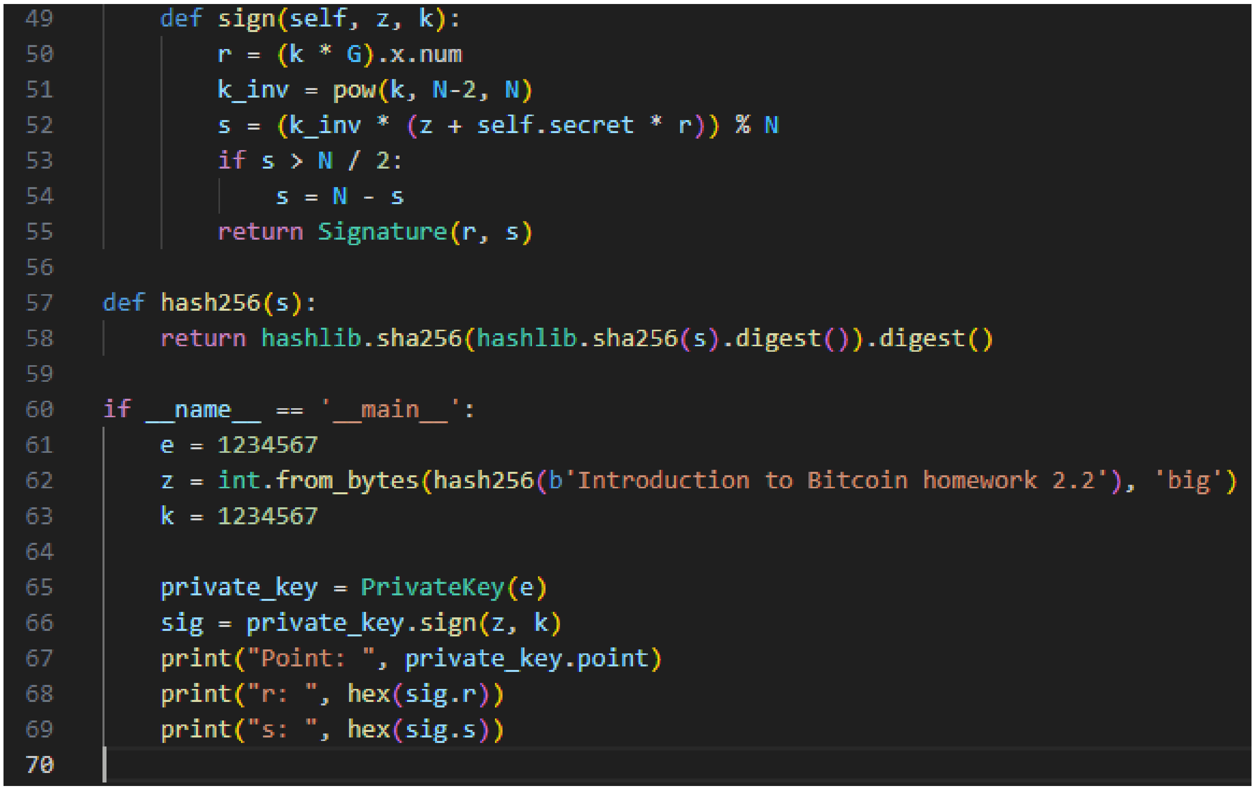
Code:

Sign.py()(包含 class S256Field, S256Point, Signature, PrivateKey，及 method hash256):

Note: 四個 class 同 problem1

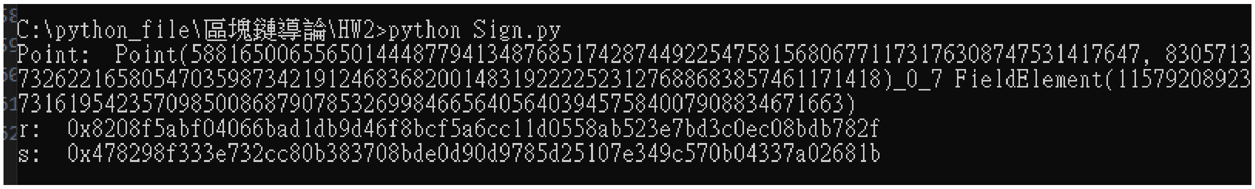


```python
from FiniteField import *
from EllipticCurves import *
import hashlib

gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
p = 2**256 - 2**32 - 977
N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
A = 0
B = 7

class S256Field(FieldElement):
    def __init__(self, num, prime=None):
        super().__init__(num = num, prime = p)

class S256Point(Point):

    def __init__(self, x, y, a=None, b=None):
        a, b = S256Field(A), S256Field(B)
        if type(x) == int:
            super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
        else: # infinity point
            super().__init__(x = x, y = y, a = a, b = b)

    def __rmul__(self, coefficient):
        coefficient = coefficient % N
        return super().__rmul__(coefficient)

    def verify(self, z, sig):
        s_inv = pow(sig.s, N-2, N)
        u = (z * s_inv) % N
        v = (sig.r * s_inv) % N
        kG = u * G + v * self
        return kG.x.num == sig.r

G = S256Point(gx, gy)


class Signature:
    def __init__(self, r, s):
        self.r = r
        self.s = s

class PrivateKey:
    def __init__(self, secret):
        self.secret  = secret
        self.point = secret * G

```



```python
    def sign(self, z, k):
        r = (k * G).x.num
        k_inv = pow(k, N-2, N)
        s = (k_inv * (z + self.secret * r)) % N
        if s > N / 2:
            s = N - s
        return Signature(r, s)

def hash256(s):
    return hashlib.sha256(hashlib.sha256(s).digest()).digest()

if __name__ == '__main__':
    e = 1234567
    z = int.from_bytes(hash256(b'Introduction to Bitcoin homework 2.2'), 'big')
    k = 1234567

    private_key = PrivateKey(e)
    sig = private_key.sign(z, k)
    print("Point: ", private_key.point)
    print("r: ", hex(sig.r))
    print("s: ", hex(sig.s))
```

Result:



```
C:\python_file\區塊鏈導論\HW2>python Sign.py
Point:  Point(58816500655650144487794134876851742874492254758156806771173176308747531417647, 8305713
73262221658054703598734219124683682001483192222523127688683857461171418)_0_7 FieldElement(11579208923
7316195423570985008687907853269984665640564039457584007908834671663)
r:  0x8208f5abf04066bad1db9d46f8bcf5a6cc11d0558ab523e7bd3c0ec08bdb782f
s:  0x478298f333e732cc80b383708bde0d90d9785d25107e349c570b04337a02681b
```

3. Solve these problems:

3-1. Find the uncompressed SEC format for the public key where the private key secrets is: 23396049

3-2. Find the compressed SEC format for the public key where the private key secrets is: 23396050

3-3. Find the DER format for a signature whose r and s values are:

r =
0x8208f5abf04066bad1db9d46f8bcf5a6cc11d0558ab523e7bd3c0ec08bdb782f

s =
0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8

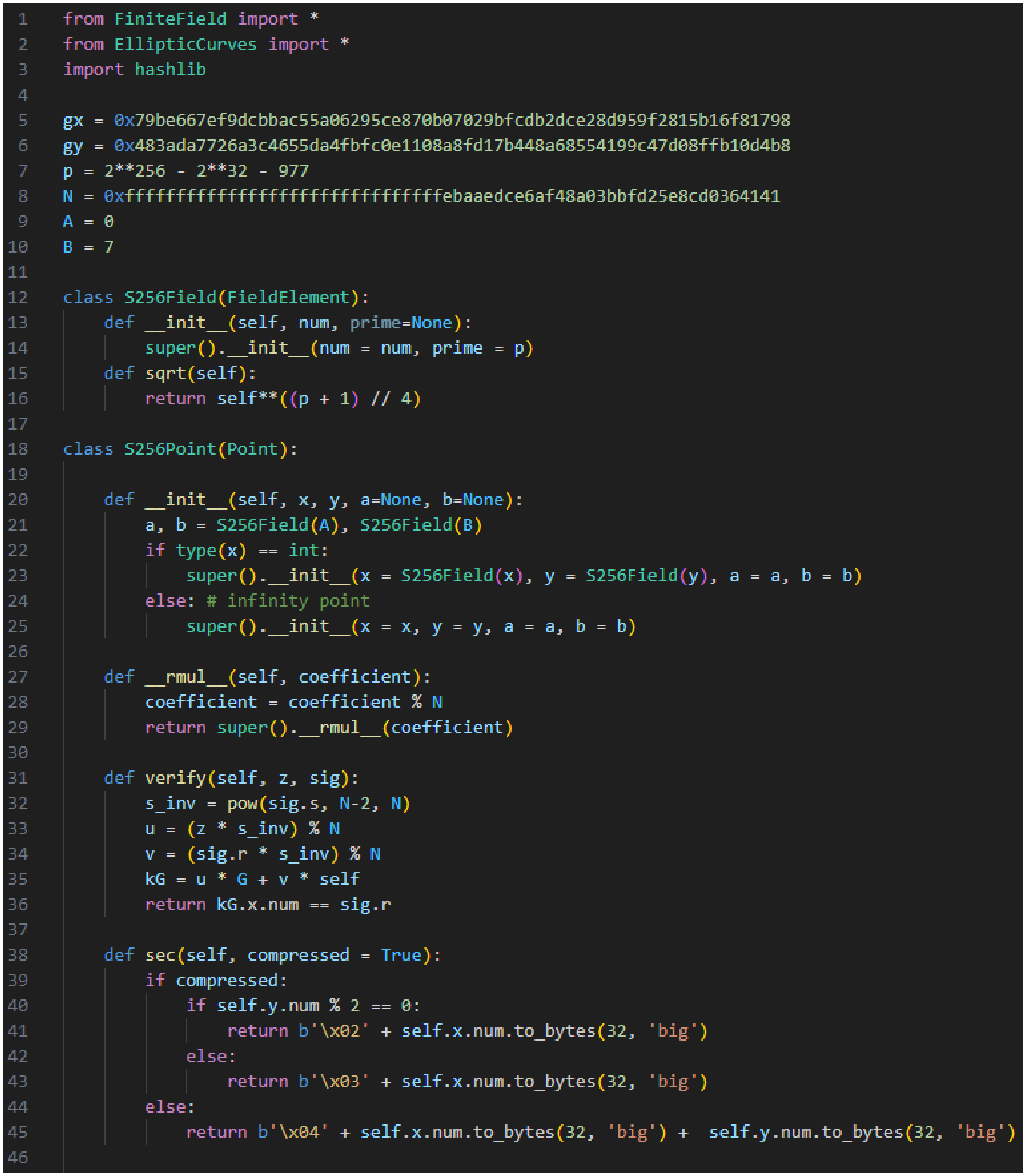
# Grading for Problem 3:

- Total: 30%
  - Correct code and code execution result: 30%, 10% for each
    (Only have correct code and correct code execution can get score.)

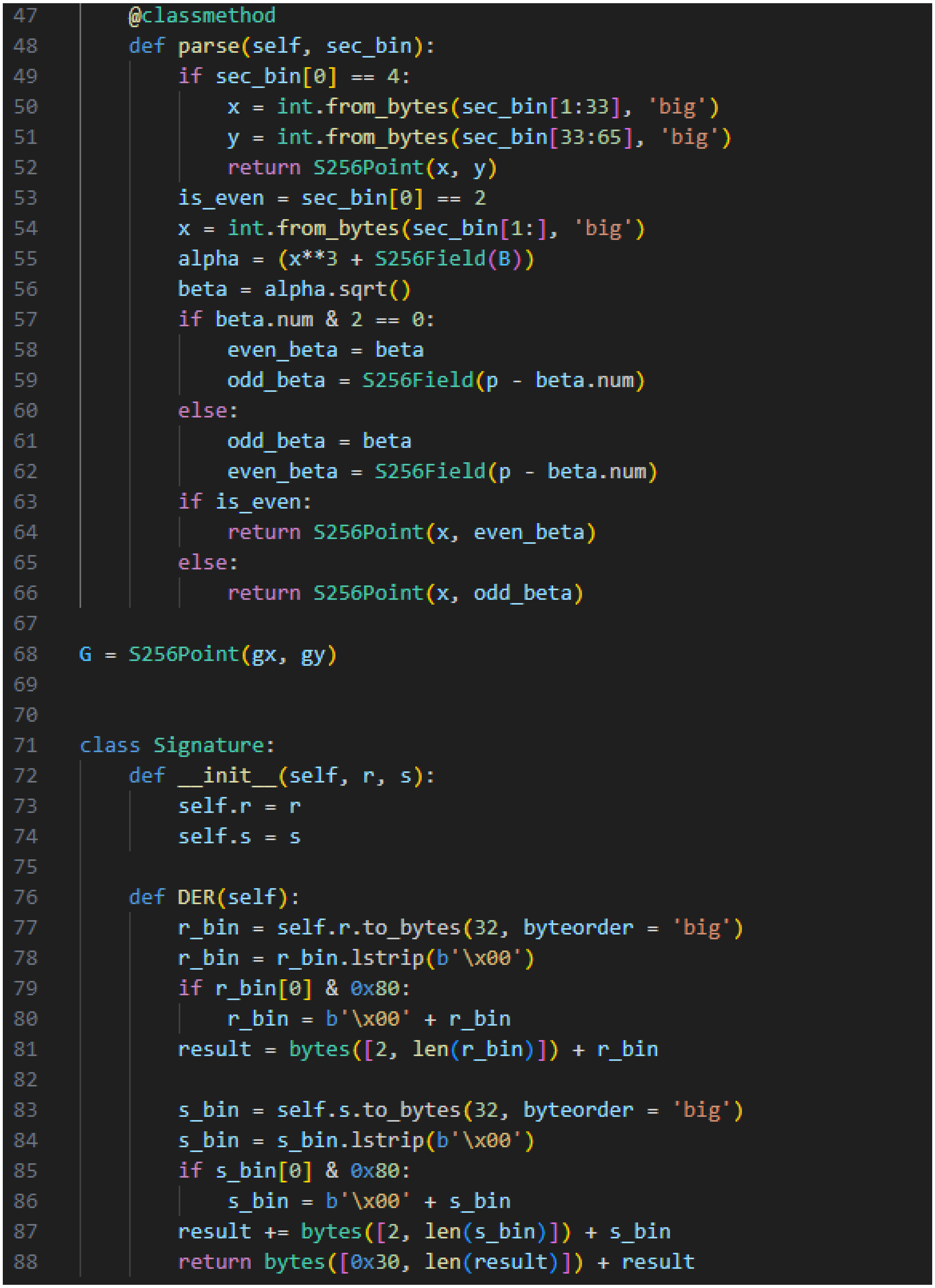
--- Please paste screenshot of your code and code execution result of solving problem 3-1 below. ---

Code:

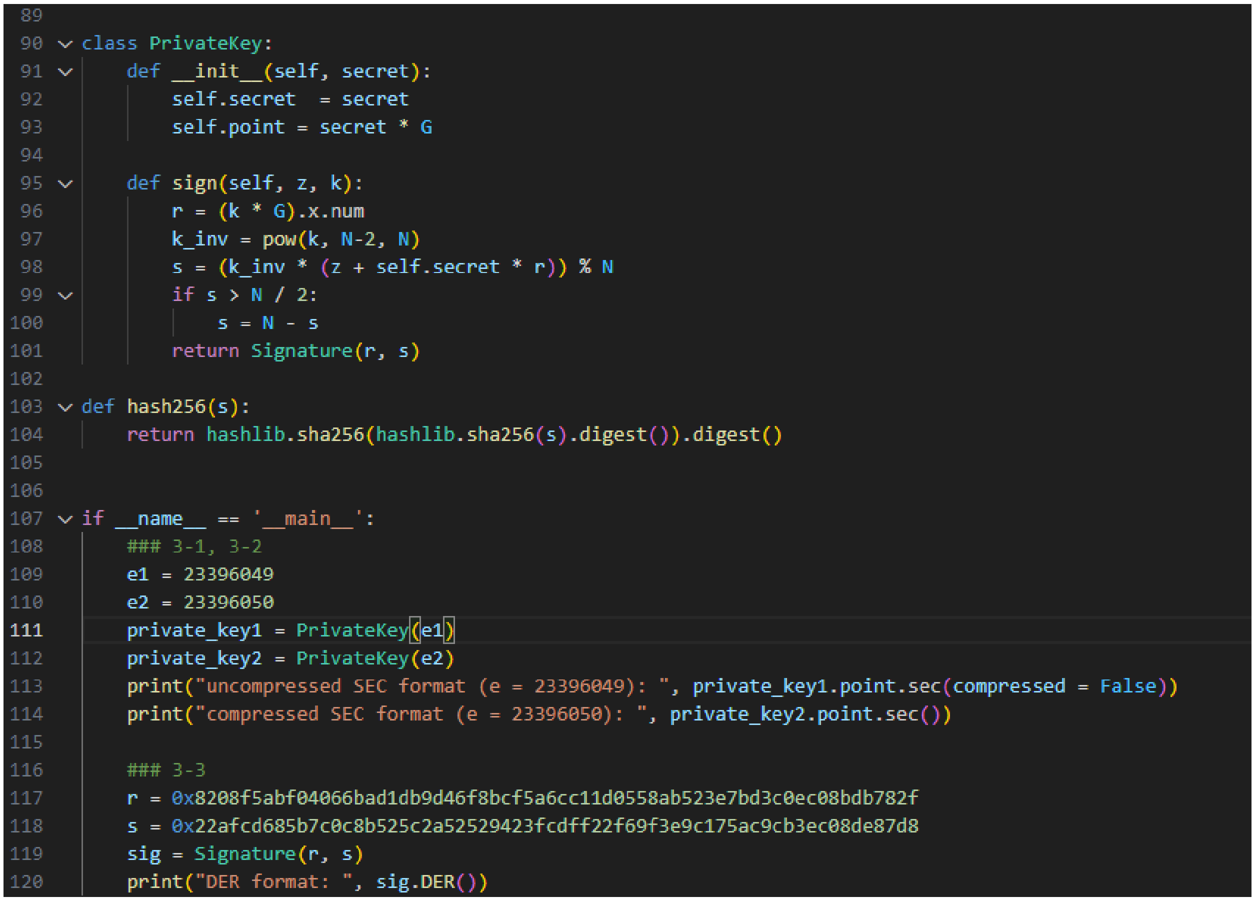
serialization.py(包含 class S256Field:新增 sqrt(), S256Point:新增 sec(), Signature: 新增 DER(), PrivateKey, 及 method hash256(), parse(),):



```
from FiniteField import *
from EllipticCurves import *
import hashlib

gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
p = 2**256 - 2**32 - 977
N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
A = 0
B = 7

class S256Field(FieldElement):
    def __init__(self, num, prime=None):
        super().__init__(num = num, prime = p)
    def sqrt(self):
        return self**((p + 1) // 4)

class S256Point(Point):

    def __init__(self, x, y, a=None, b=None):
        a, b = S256Field(A), S256Field(B)
        if type(x) == int:
            super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
        else: # infinity point
            super().__init__(x = x, y = y, a = a, b = b)

    def __rmul__(self, coefficient):
        coefficient = coefficient % N
        return super().__rmul__(coefficient)

    def verify(self, z, sig):
        s_inv = pow(sig.s, N-2, N)
        u = (z * s_inv) % N
        v = (sig.r * s_inv) % N
        kG = u * G + v * self
        return kG.x.num == sig.r

    def sec(self, compressed = True):
        if compressed:
            if self.y.num % 2 == 0:
                return b'\x02' + self.x.num.to_bytes(32, 'big')
            else:
                return b'\x03' + self.x.num.to_bytes(32, 'big')
        else:
            return b'\x04' + self.x.num.to_bytes(32, 'big') +  self.y.num.to_bytes(32, 'big')

```

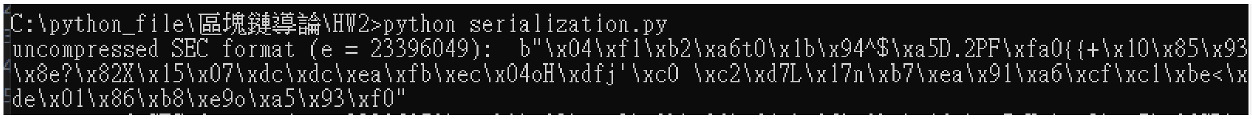


```python
    @classmethod
    def parse(self, sec_bin):
        if sec_bin[0] == 4:
            x = int.from_bytes(sec_bin[1:33], 'big')
            y = int.from_bytes(sec_bin[33:65], 'big')
            return S256Point(x, y)
        is_even = sec_bin[0] == 2
        x = int.from_bytes(sec_bin[1:], 'big')
        alpha = (x**3 + S256Field(B))
        beta = alpha.sqrt()
        if beta.num & 2 == 0:
            even_beta = beta
            odd_beta = S256Field(p - beta.num)
        else:
            odd_beta = beta
            even_beta = S256Field(p - beta.num)
        if is_even:
            return S256Point(x, even_beta)
        else:
            return S256Point(x, odd_beta)

G = S256Point(gx, gy)


class Signature:
    def __init__(self, r, s):
        self.r = r
        self.s = s

    def DER(self):
        r_bin = self.r.to_bytes(32, byteorder = 'big')
        r_bin = r_bin.lstrip(b'\x00')
        if r_bin[0] & 0x80:
            r_bin = b'\x00' + r_bin
        result = bytes([2, len(r_bin)]) + r_bin

        s_bin = self.s.to_bytes(32, byteorder = 'big')
        s_bin = s_bin.lstrip(b'\x00')
        if s_bin[0] & 0x80:
            s_bin = b'\x00' + s_bin
        result += bytes([2, len(s_bin)]) + s_bin
        return bytes([0x30, len(result)]) + result
```



```python

class PrivateKey:
    def __init__(self, secret):
        self.secret  = secret
        self.point = secret * G

    def sign(self, z, k):
        r = (k * G).x.num
        k_inv = pow(k, N-2, N)
        s = (k_inv * (z + self.secret * r)) % N
        if s > N / 2:
            s = N - s
        return Signature(r, s)

def hash256(s):
    return hashlib.sha256(hashlib.sha256(s).digest()).digest()


if __name__ == '__main__':
    ### 3-1, 3-2
    e1 = 23396049
    e2 = 23396050
    private_key1 = PrivateKey(e1)
    private_key2 = PrivateKey(e2)
    print("uncompressed SEC format (e = 23396049): ", private_key1.point.sec(compressed = False))
    print("compressed SEC format (e = 23396050): ", private_key2.point.sec())

    ### 3-3
    r = 0x8208f5abf04066bad1db9d46f8bcf5a6cc11d0558ab523e7bd3c0ec08bdb782f
    s = 0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8
    sig = Signature(r, s)
    print("DER format: ", sig.DER())
```

Result:



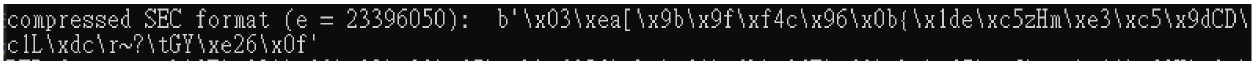
```
C:\python_file\區塊鏈導論\HW2>python serialization.py
uncompressed SEC format (e = 23396049):  b"\x04\xf1\xb2\xa6t0\x1b\x94^$\xa5D.2PF\xfa0{{+\x10\x85\x93
\x8e?\x82X\x15\x07\xdc\xdc\xea\xfb\xec\x04oH\xdfj'\xc0 \xc2\xd7L\x17n\xb7\xea\x91\xa6\xcf\xc1\xbe<\x
de\x01\x86\xb8\xe9o\xa5\x93\xf0"
```

--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 3-2 below. ---

Code:

同 problem 3-1

Result:



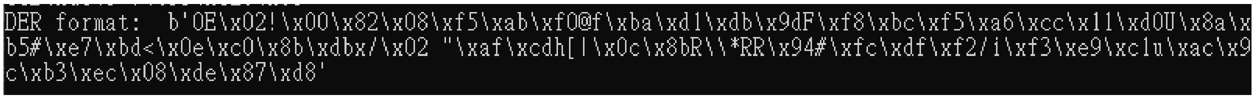
```
compressed SEC format (e = 23396050):  b'\x03\xea[\x9b\x9f\xf4c\x96\x0b{\x1de\xc5zHm\xe3\xc5\x9dCD\
c1L\xdc\r~?\tGY\xe26\x0f'
```

--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 3-3 below. ---

Code:

同 problem 3-1

Result:



```
DER format:  b'0E\x02!\x00\x82\x08\xf5\xab\xf0@f\xba\xd1\xdb\x9dF\xf8\xbc\xf5\xa6\xcc\x11\xd0U\x8a\x
b5#\xe7\xbd<\x0e\xc0\x8b\xdbx/\x02 "\xaf\xcdh[|\x0c\x8bR\\*RR\x94#\xfc\xdf\xf2/i\xf3\xe9\xc1u\xac\x9
c\xb3\xec\x08\xde\x87\xd8'
```

4. Compute the slope and the sum of the points:

4-1. Find the address corresponding to Public Keys whose Private Key secrets are:

23396051 (use uncompressed SEC, on testnet) (10%)

23396052 (use compressed SEC, on testnet) (10%)

4-2. Find the WIF for Private Key whose the secret is:

23396053 (use compressed SEC, on testnet) (10%)

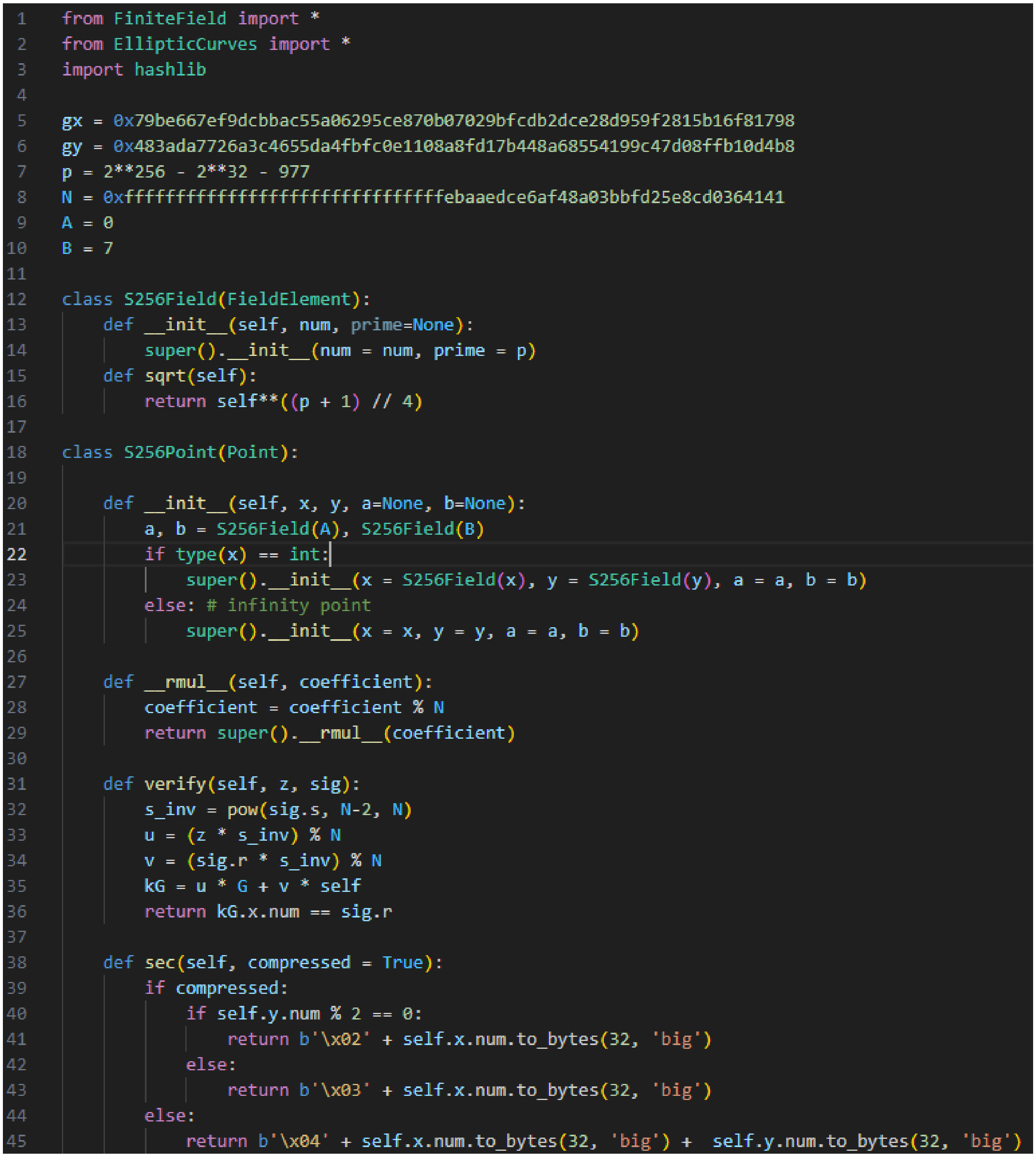
# Grading for Problem 4:

- Total: 30%
  - Correct code and code execution result: 30%, 10% for each
    (Only have correct code and correct code execution can get score.)

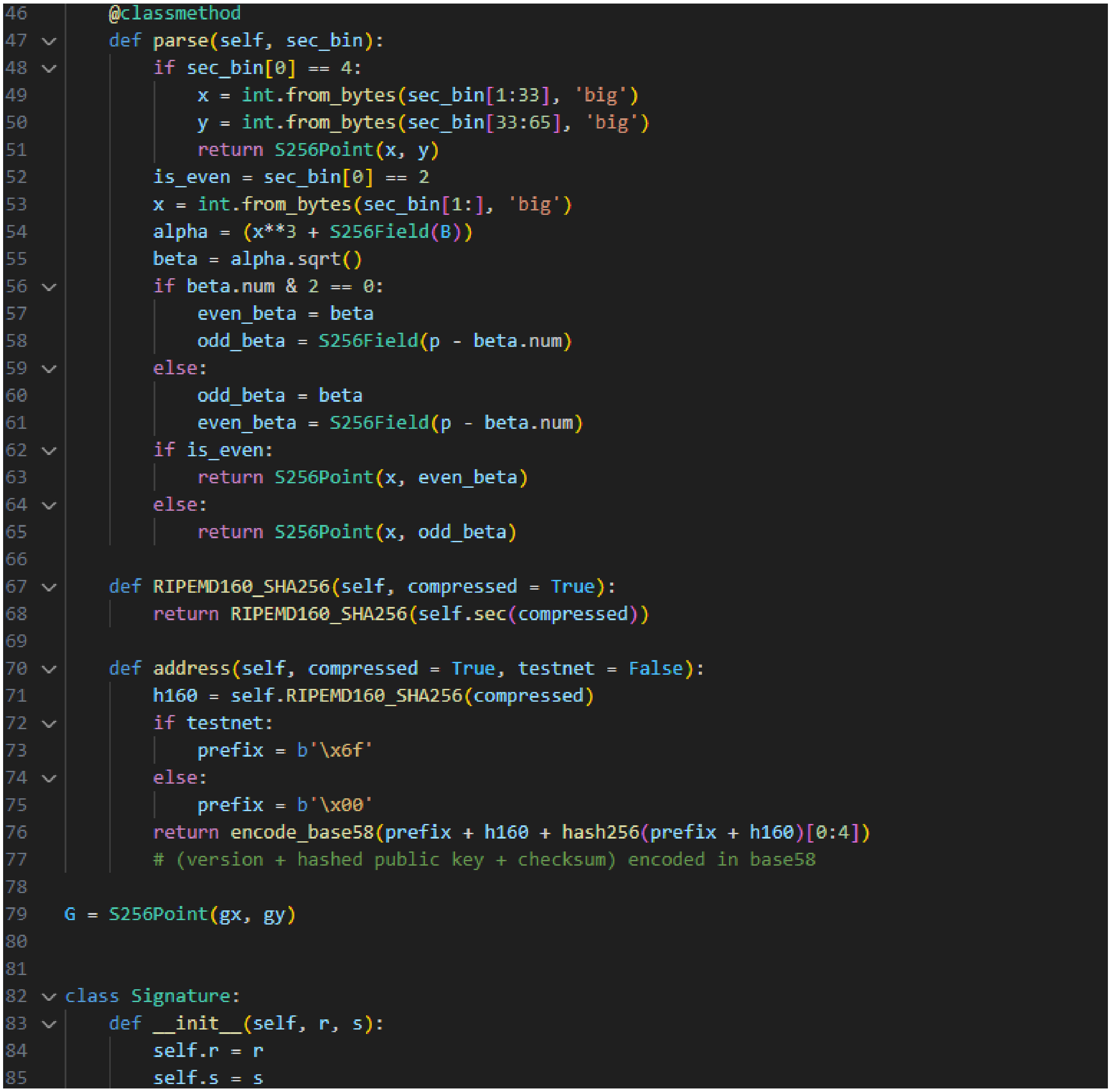
--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 4-1 below. ---

Code:

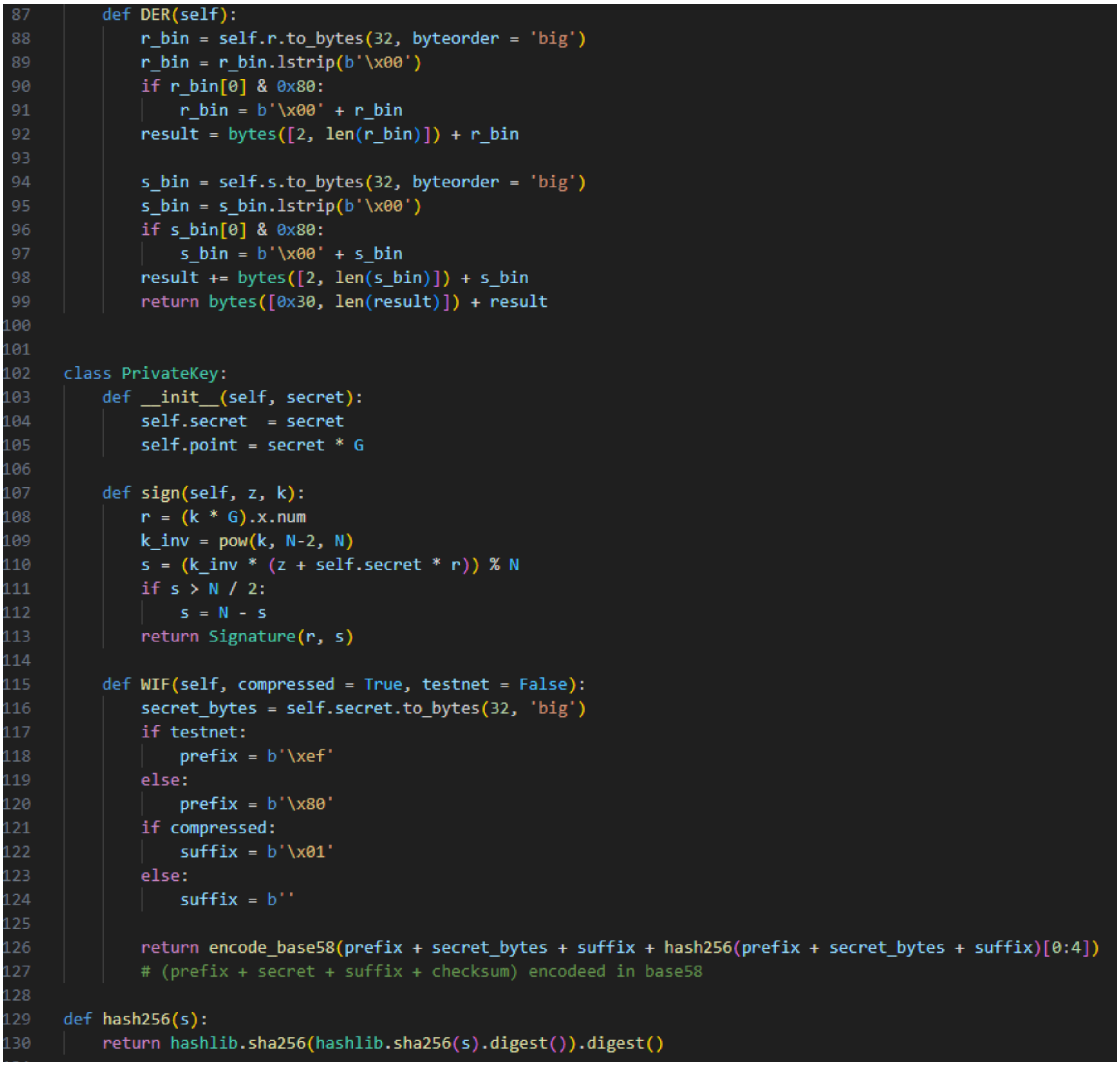
Address_and_WIF.py(包含 S256Field, S256Point: 新增 RIPEMD160_SHA256()及 address(), Signature, PrivateKey: 新增 WIF(), 及 method hash256, parse, encode_base58(), RIPEMD160_SHA256()):



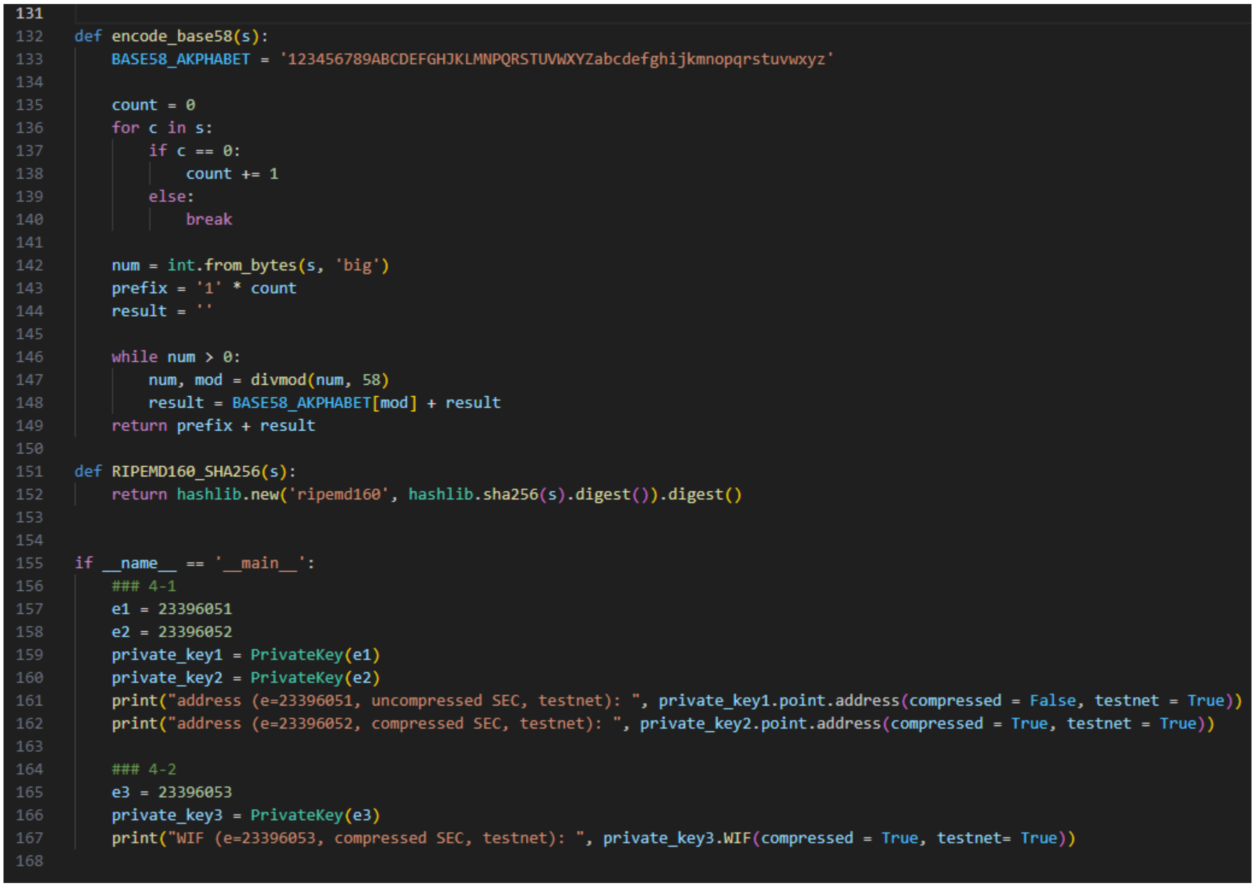
```python
from FiniteField import *
from EllipticCurves import *
import hashlib

gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
p = 2**256 - 2**32 - 977
N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
A = 0
B = 7

class S256Field(FieldElement):
    def __init__(self, num, prime=None):
        super().__init__(num = num, prime = p)
    def sqrt(self):
        return self**((p + 1) // 4)

class S256Point(Point):

    def __init__(self, x, y, a=None, b=None):
        a, b = S256Field(A), S256Field(B)
        if type(x) == int:
            super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
        else: # infinity point
            super().__init__(x = x, y = y, a = a, b = b)

    def __rmul__(self, coefficient):
        coefficient = coefficient % N
        return super().__rmul__(coefficient)

    def verify(self, z, sig):
        s_inv = pow(sig.s, N-2, N)
        u = (z * s_inv) % N
        v = (sig.r * s_inv) % N
        kG = u * G + v * self
        return kG.x.num == sig.r

    def sec(self, compressed = True):
        if compressed:
            if self.y.num % 2 == 0:
                return b'\x02' + self.x.num.to_bytes(32, 'big')
            else:
                return b'\x03' + self.x.num.to_bytes(32, 'big')
        else:
            return b'\x04' + self.x.num.to_bytes(32, 'big') +  self.y.num.to_bytes(32, 'big')
```



```python
    @classmethod
    def parse(self, sec_bin):
        if sec_bin[0] == 4:
            x = int.from_bytes(sec_bin[1:33], 'big')
            y = int.from_bytes(sec_bin[33:65], 'big')
            return S256Point(x, y)
        is_even = sec_bin[0] == 2
        x = int.from_bytes(sec_bin[1:], 'big')
        alpha = (x**3 + S256Field(B))
        beta = alpha.sqrt()
        if beta.num & 2 == 0:
            even_beta = beta
            odd_beta = S256Field(p - beta.num)
        else:
            odd_beta = beta
            even_beta = S256Field(p - beta.num)
        if is_even:
            return S256Point(x, even_beta)
        else:
            return S256Point(x, odd_beta)

    def RIPEMD160_SHA256(self, compressed = True):
        return RIPEMD160_SHA256(self.sec(compressed))

    def address(self, compressed = True, testnet = False):
        h160 = self.RIPEMD160_SHA256(compressed)
        if testnet:
            prefix = b'\x6f'
        else:
            prefix = b'\x00'
        return encode_base58(prefix + h160 + hash256(prefix + h160)[0:4])
        # (version + hashed public key + checksum) encoded in base58

G = S256Point(gx, gy)


class Signature:
    def __init__(self, r, s):
        self.r = r
        self.s = s
```

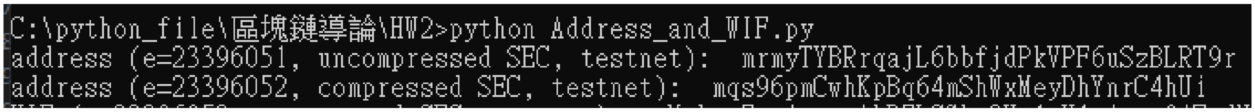


```python
    def DER(self):
        r_bin = self.r.to_bytes(32, byteorder = 'big')
        r_bin = r_bin.lstrip(b'\x00')
        if r_bin[0] & 0x80:
            r_bin = b'\x00' + r_bin
        result = bytes([2, len(r_bin)]) + r_bin

        s_bin = self.s.to_bytes(32, byteorder = 'big')
        s_bin = s_bin.lstrip(b'\x00')
        if s_bin[0] & 0x80:
            s_bin = b'\x00' + s_bin
        result += bytes([2, len(s_bin)]) + s_bin
        return bytes([0x30, len(result)]) + result


class PrivateKey:
    def __init__(self, secret):
        self.secret  = secret
        self.point = secret * G

    def sign(self, z, k):
        r = (k * G).x.num
        k_inv = pow(k, N-2, N)
        s = (k_inv * (z + self.secret * r)) % N
        if s > N / 2:
            s = N - s
        return Signature(r, s)

    def WIF(self, compressed = True, testnet = False):
        secret_bytes = self.secret.to_bytes(32, 'big')
        if testnet:
            prefix = b'\xef'
        else:
            prefix = b'\x80'
        if compressed:
            suffix = b'\x01'
        else:
            suffix = b''

        return encode_base58(prefix + secret_bytes + suffix + hash256(prefix + secret_bytes + suffix)[0:4])
        # (prefix + secret + suffix + checksum) encodeed in base58

def hash256(s):
    return hashlib.sha256(hashlib.sha256(s).digest()).digest()
```



```python

def encode_base58(s):
    BASE58_AKPHABET = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'

    count = 0
    for c in s:
        if c == 0:
            count += 1
        else:
            break

    num = int.from_bytes(s, 'big')
    prefix = '1' * count
    result = ''

    while num > 0:
        num, mod = divmod(num, 58)
        result = BASE58_AKPHABET[mod] + result
    return prefix + result

def RIPEMD160_SHA256(s):
    return hashlib.new('ripemd160', hashlib.sha256(s).digest()).digest()


if __name__ == '__main__':
    ### 4-1
    e1 = 23396051
    e2 = 23396052
    private_key1 = PrivateKey(e1)
    private_key2 = PrivateKey(e2)
    print("address (e=23396051, uncompressed SEC, testnet): ", private_key1.point.address(compressed = False, testnet = True))
    print("address (e=23396052, compressed SEC, testnet): ", private_key2.point.address(compressed = True, testnet = True))

    ### 4-2
    e3 = 23396053
    private_key3 = PrivateKey(e3)
    print("WIF (e=23396053, compressed SEC, testnet): ", private_key3.WIF(compressed = True, testnet= True))

```

Result:



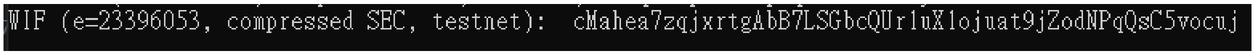
```
C:\python_file\區塊鏈導論\HW2>python Address_and_WIF.py
address (e=23396051, uncompressed SEC, testnet):  mrmyTYBRrqajL6bbfjdPkVPF6uSzBLRT9r
address (e=23396052, compressed SEC, testnet):  mqs96pmCwhKpBq64mShWxMeyDhYnrC4hUi
```

--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 4-2 below. ---

Code:

同 problem 4-1

Result:



```
WIF (e=23396053, compressed SEC, testnet):  cMahea7zqjxrtgAbB7LSGbcQUr1uX1ojuat9jZodNPqQsC5vocuj
```