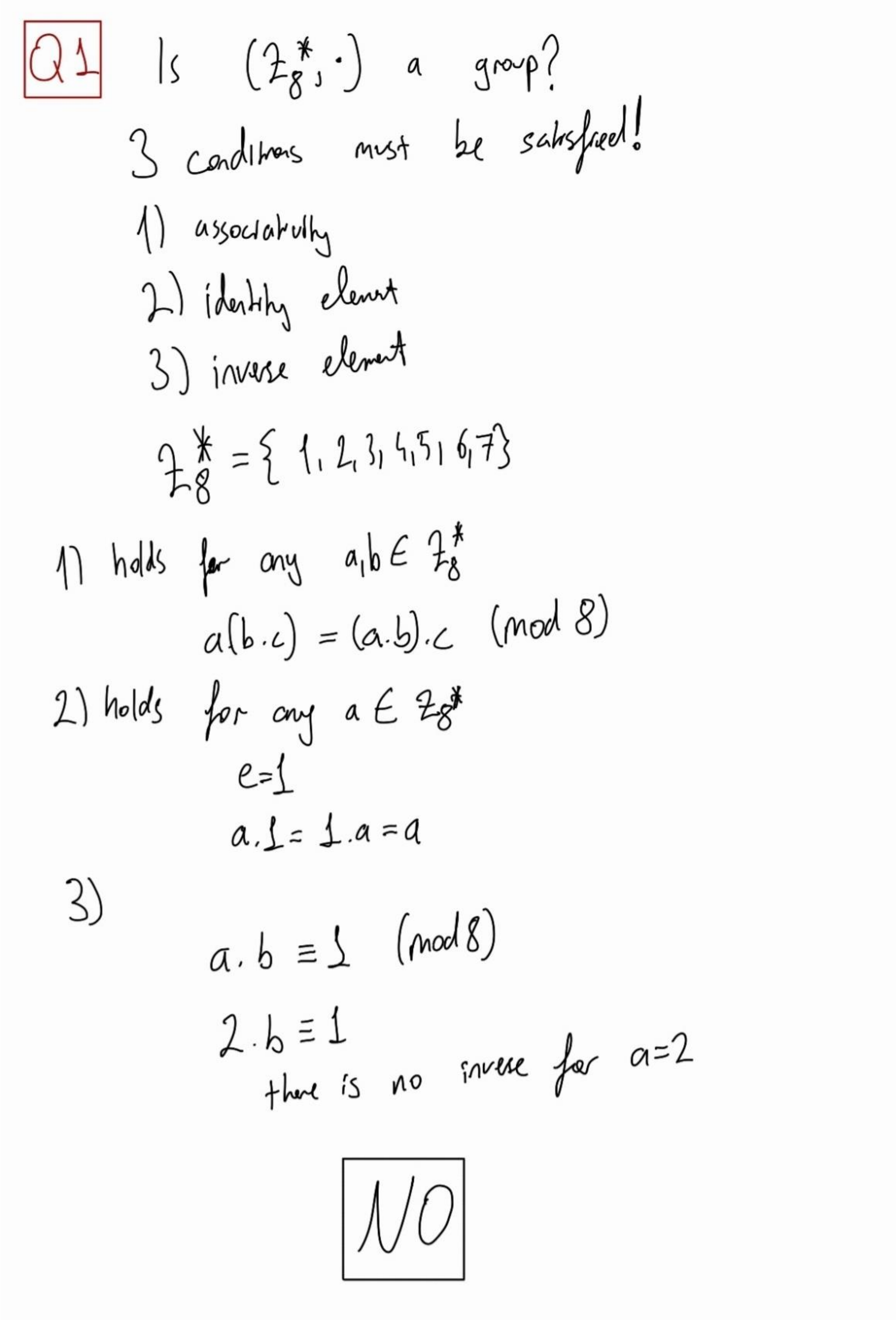
Ozan Cinci

2448223

HW1



$\boxed{Q1}$ Is $(\mathbb{Z}_8^*, \cdot)$ a group?

3 conditions must be satisfied!

1) associativity
2) identity element
3) inverse element

$\mathbb{Z}_8^* = \{1, 2, 3, 4, 5, 6, 7\}$

1) holds for any $a, b \in \mathbb{Z}_8^*$

$$a(b.c) = (a.b).c \pmod 8$$

2) holds for any $a \in \mathbb{Z}_8^*$

$$e = 1$$

$$a.1 = 1.a = a$$

3)

$$a.b \equiv 1 \pmod 8$$

$$2.b \equiv 1$$

there is no inverse for $a=2$

$$\boxed{NO}$$

Q2 Provide a generator for $(\mathbb{Z}_{13}^*, \cdot)$. Show it generates every element in $\mathbb{Z}_{13}^*$.

$$\mathbb{Z}_{13}^* = \{1, 2, 3, \dots 12\}$$

$2^0 \equiv 1$

$2^1 \equiv 2$

$2^2 \equiv 4$

$2^3 \equiv 8$

$2^4 \equiv 3$

$2^5 \equiv 6$

$2^6 \equiv 12$

$2^7 \equiv 11$

$2^8 \equiv 9$

$2^9 \equiv 5$

$2^{10} \equiv 10$

$2^{11} \equiv 7$

$2^{12} \equiv 1$

Order of 2 is 12 which proves that it is a generator for this group!

$\boxed{2}$

Q3

E : $y^2 = x^3 + x + 3$ over F_{13}

1) $y^2 = x^3 + ax + b$ must be the format

$a = 1$ and $b = 3$ holds the requirement

2) $q = 13$ is prime

3) $4a^2 + 27b^3 \not\equiv 0$ mod 13 must holds

$a = 1$
$b = 3$

$4.1 + 27.27 \equiv 5 \pmod{13}$

$4a^2 + 27b^3 \not\equiv 0$ holds

All the 3 conditions are met.

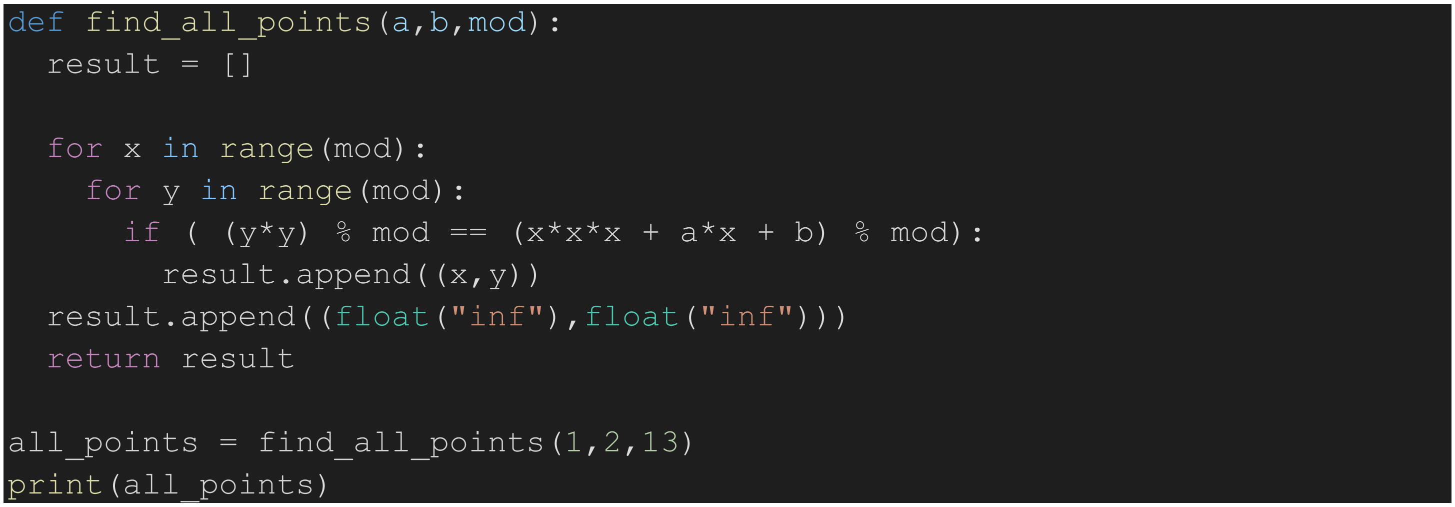
$\boxed{\text{Yes}}$

Q4.

It has 12 points, which are:

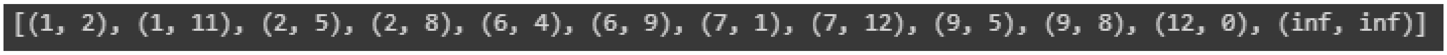
[(1, 2), (1, 11), (2, 5), (2, 8), (6, 4), (6, 9), (7, 1), (7, 12), (9, 5), (9, 8), (12, 0), (inf, inf)]

find_all_points(a,b,mod) is a function that finds all the possible points on elliptic curve. It looks through all possible x and y values in mod 13. Neutral element 0 which is represented as (float("inf"), float("inf")). This function returns array of integer.



```
def find_all_points(a,b,mod):
  result = []

  for x in range(mod):
    for y in range(mod):
      if ( (y*y) % mod == (x*x*x + a*x + b) % mod):
        result.append((x,y))
  result.append((float("inf"),float("inf")))
  return result

all_points = find_all_points(1,2,13)
print(all_points)
```

result:

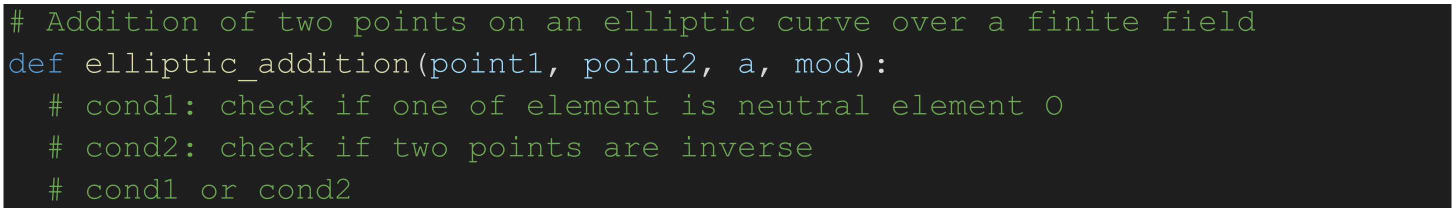


```
[(1, 2), (1, 11), (2, 5), (2, 8), (6, 4), (6, 9), (7, 1), (7, 12), (9, 5), (9, 8), (12, 0), (inf, inf)]
```

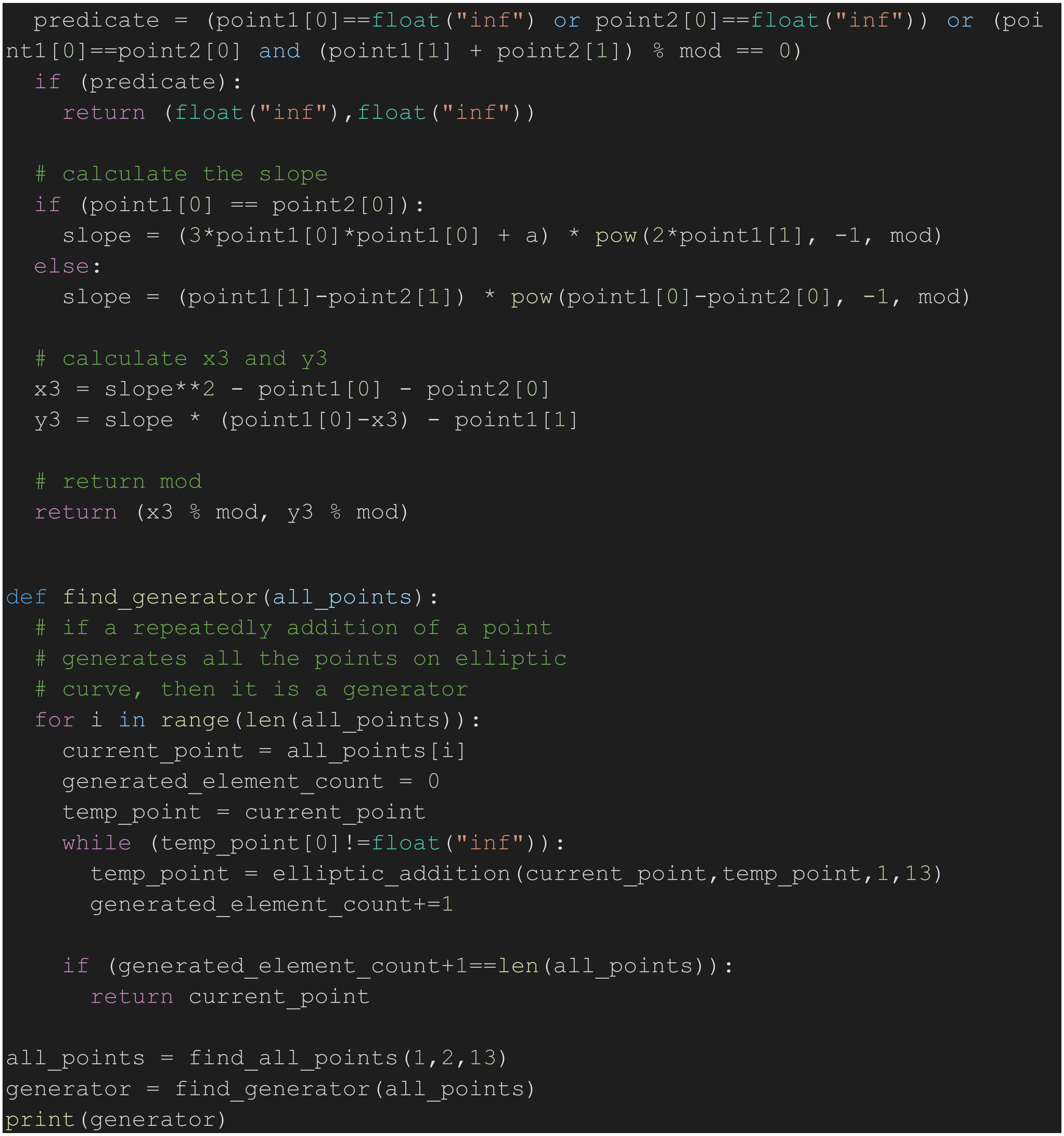
Q5.

Generator shall produce all the points if we repeatedly calculate cumulation. Let's say we have generator P. Then when we calculate P, 2P, 3P, 4P, 5P…… we would calculate all possible points. We know all the possible points from previous questions answer. Then we should check every point in elliptic curve array to see if it holds mentioned property. If it holds, then it is a generator.

I have implemented a point calculation function. That calculates the result of addition of two points on elliptic curve. The calculated all possible cumulative additions for a point. If #points generated is equal to the #points on elliptic curve, then it is a generator. After running my code, the result was (6,4).

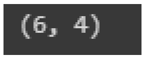


```
# Addition of two points on an elliptic curve over a finite field
def elliptic_addition(point1, point2, a, mod):
  # cond1: check if one of element is neutral element O
  # cond2: check if two points are inverse
  # cond1 or cond2
```



```
  predicate = (point1[0]==float("inf") or point2[0]==float("inf")) or (poi
nt1[0]==point2[0] and (point1[1] + point2[1]) % mod == 0)
  if (predicate):
    return (float("inf"),float("inf"))

  # calculate the slope
  if (point1[0] == point2[0]):
    slope = (3*point1[0]*point1[0] + a) * pow(2*point1[1], -1, mod)
  else:
    slope = (point1[1]-point2[1]) * pow(point1[0]-point2[0], -1, mod)

  # calculate x3 and y3
  x3 = slope**2 - point1[0] - point2[0]
  y3 = slope * (point1[0]-x3) - point1[1]

  # return mod
  return (x3 % mod, y3 % mod)


def find_generator(all_points):
  # if a repeatedly addition of a point
  # generates all the points on elliptic
  # curve, then it is a generator
  for i in range(len(all_points)):
    current_point = all_points[i]
    generated_element_count = 0
    temp_point = current_point
    while (temp_point[0]!=float("inf")):
      temp_point = elliptic_addition(current_point,temp_point,1,13)
      generated_element_count+=1

    if (generated_element_count+1==len(all_points)):
      return current_point

all_points = find_all_points(1,2,13)
generator = find_generator(all_points)
print(generator)
```

result:



```
(6, 4)
```