The following code was submitted towards my Python 6850 class as the final project, the code ended up receiving 100% on the assignment.

```
"""
Code requirements I made for myself
```

```
create a python project to tell a user-given IP address's network address,
address range, broadcast, and next address in line
also prompt for subnet mask in slash notation, eg: /24
subnets have to be between 0 and 31
IP decimal octets must be between 0 and 255
store answers in database for later access to check answers
create its own inputs if errors occur or if the code is attempted to be broken
use https://subnetipv4.com/ to test answers.
The website isn't always perfect, though, so those I needed to double check I
wrote by hand
```

```
What is subnetting?
```

```
Rather than explaining the history of subnets and how they work, I decided I'd
just paste a link to an article explaining it,
because the assignment is the code, not the history of subnets
https://bit.ly/31TlqFT
Essentially it is a series of calculations given a certain ip address and
subnet mask
```

```
How this code works
```

```
1. It asks for an ip address from the user (it makes its own if the user
enters incorrectly)
2. It asks for a subnet mask and makes its own, similar to step 1.
3. A series of calculations for the Network address, First Host address, Last
Host address, Broadcast address, and the Next subnet
4. It stores these values in a database for withdrawal in case a repeated ip
address is entered
```

```
Why did I pick this project?
```

```
I am taking three classes this semester, Networking & Servers, Database Theory
& Design, and Python.
During the networking class, they introduced a method to calculate subnets by
hand.
```

I wanted to practice subnetting, so I found a website that could make subnets for me.
Halfway through I realized I could get some Python practice in if I made a code that could make subnets.
This project comes around as a culmination of each of my classes to help me learn better in each.

```
"""
import sqlite3
import random


def database_connect():
    # This is a very simple function to just start the database at the start of the code
    # This code essentially opens and closes the database for a fresh start when the database is accessed later
    dbase = sqlite3.connect('subnet_db_001.db', timeout=5)
    dtb = dbase.cursor()
    dbase.commit()
    # This command appears later in the code, it creates the database in case it's not already created
    dtb.execute("""CREATE TABLE IF NOT EXISTS "ip_address" (
                "ip" TEXT,
                "subnet" TEXT,
                "network" TEXT,
                "first_host" TEXT,
                "last_host" TEXT,
                "broadcast" TEXT,
                "next_subnet" INTEGER,
                PRIMARY KEY("ip")
                );""")
    dbase.commit()
    dbase.close()


def ipv4_number_format(num):
    # The ipv4 function ensures the numbers are under 256
    # Essentially numbers under 0 are 255 and those over 255 are 0
    num = int(num)
    if num > 255:
        num = 0
        return num
    elif num < 0:
        num = 255
        return num
    else:
        return num
```

```
def ensure_magic_number_is_less_than_octet_of_interest(mn, oi):
    #                     Where x = magic number and z = octet of interest
    # use this function to ensure the magic number doesn't make any number
higher than the octet of interest
    x = int(mn)
    z = int(oi)
    y = 0
    while x * y < z:
        # This while loop calculates what the closest multiple of 'mn' can be
while still being less than 'oi'
        y += 1
    if x * y == z:
        # This if statement ensures that if x*y is z, then end the function
        pass
    else:
        # This else statement keeps y * x under z, if they match it should be
okay, I can't find any solid sources to confirm
        y -= 1
    if y < 0:
        # This if statement is to make sure y does not end up being negative
        y = 0
    #      Multiply this number by the magic number
    return int(y)


def next_sub_calculator(oct1, oct2, oct3, next_subnet):
    # This calculates the next subnet address and ensures that the other octets
increase if necessary
    i1 = int(oct1)
    i2 = int(oct2)
    i3 = int(oct3)
    n_subnet = int(next_subnet)
    # Each if statement in this function increases the next inputted digit by 1
if needed
    if n_subnet > 255:
        i3 += 1
        n_subnet = ipv4_number_format(n_subnet)
    if i3 > 255:
        i2 += 1
        i3 = ipv4_number_format(i3)
    if i2 > 255:
        i1 += 1
        i2 = ipv4_number_format(i2)
    if i1 > 255:
        i1 = ipv4_number_format(i1)
    return i1, i2, i3, n_subnet
```

```
def database_mine(input1, input2, input3, input4, subnet):
    # This function is to search the database for the ip address entered and
return it if it has it, otherwise, the code runs as normal
    # This code opens the database and creates the needed table if it doesn't
exist
    database = sqlite3.connect('subnet_db_001.db', timeout=5)
    db = database.cursor()
    db.execute("""CREATE TABLE IF NOT EXISTS "ip_address" (
            "ip" TEXT,
            "subnet" TEXT,
            "network" TEXT,
            "first_host" TEXT,
            "last_host" TEXT,
            "broadcast" TEXT,
            "next_subnet" INTEGER,
            PRIMARY KEY("ip")
            );""")
    # This list stores each answer from the database that will later be
returned as the answers
    ip_answers = []
    # This decision and if statement checks if the input is in the database
    decision = db.execute(f"SELECT DISTINCT ip FROM ip_address "
                          f"WHERE ip = '{input1}.{input2}.{input3}.{input4}'
AND subnet = '{subnet}'").fetchone()
    if decision is None:
        # This ends the function if True
        return None
    else:
        # This will continue the function
        pass
    # Each of these select/fetch sql statements are added to the list:
ip_answers
    ip_db_network = db.execute(f"SELECT DISTINCT network FROM ip_address "
                               f"WHERE ip =
'{input1}.{input2}.{input3}.{input4}' AND subnet = '{subnet}';").fetchone()
    ip_db_first = db.execute(f"SELECT DISTINCT first_host FROM ip_address "
                             f"WHERE ip = '{input1}.{input2}.{input3}.{input4}'
AND subnet = '{subnet}';").fetchone()
    ip_db_last = db.execute(f"SELECT DISTINCT last_host FROM ip_address "
                            f"WHERE ip = '{input1}.{input2}.{input3}.{input4}'
AND subnet = '{subnet}';").fetchone()
    ip_db_broadcast = db.execute(f"SELECT DISTINCT broadcast FROM ip_address "
                                 f"WHERE ip =
'{input1}.{input2}.{input3}.{input4}' AND subnet = '{subnet}';").fetchone()
    ip_db_next = db.execute(f"SELECT DISTINCT next_subnet FROM ip_address "
                            f"WHERE ip = '{input1}.{input2}.{input3}.{input4}'
AND subnet = '{subnet}';").fetchone()
    # add each above variables to the list: ip_answers
```

```
    # 0
    ip_answers.append(ip_db_network)
    # 1
    ip_answers.append(ip_db_first)
    # 2
    ip_answers.append(ip_db_last)
    # 3
    ip_answers.append(ip_db_broadcast)
    # 4
    ip_answers.append(ip_db_next)
    # without these replace/string statements, each item in the list is a
tuple, I don't need tuples, I need strings
    network_address = str(ip_answers[0]).replace("(", "").replace("'",
"").replace(")", "").replace(",", "")
    first_host_address = str(ip_answers[1]).replace("(", "").replace("'",
"").replace(")", "").replace(",", "")
    last_host_address = str(ip_answers[2]).replace("(", "").replace("'",
"").replace(")", "").replace(",", "")
    broadcast_address = str(ip_answers[3]).replace("(", "").replace("'",
"").replace(")", "").replace(",", "")
    next_subnet_address = str(ip_answers[4]).replace("(", "").replace("'",
"").replace(")", "").replace(",", "")
    # closes the database
    database.commit()
    database.close()
    # This returns the answers in a similar format to the PrintAnswer class
    return f"Network Address:              {network_address}\n" \
           f"First Host Address:           {first_host_address}\n" \
           f"Last Host Address:            {last_host_address}\n" \
           f"Broadcast Address:            {broadcast_address}\n" \
           f"Next Subnet:                  {next_subnet_address}"


class MagicNumber:
    #           This class and it's subclasses are for calculating the magic
number for use in all of the calculations
    # self.number is the inputted subnet
    def __init__(self, number):
        self.number = int(number)


class ClassA(MagicNumber):
    # These classes are not official ip terminology, they are my own
classification to make coding easier
    m = 32

    def magic(self):
        # class A = 2^(32-subnet)
        power = self.m - self.number
```

```
        magic = 2 ** power
        return int(magic)


class ClassB(MagicNumber):
    m = 24

    def magic(self):
        # class B = 2^(24-subnet)
        power = self.m - self.number
        magic = 2 ** power
        return int(magic)


class ClassC(MagicNumber):
    m = 16

    def magic(self):
        # class C = 2^(16-subnet)
        power = self.m - self.number
        magic = 2 ** power
        return int(magic)


class ClassD(MagicNumber):
    m = 8

    def magic(self):
        # class D = 2^(8-subnet)
        power = self.m - self.number
        magic = 2 ** power
        return int(magic)


class PrintAnswer:
    #                   This class is used for printing the answer in the
proper format depending on the subnet class
    # Most of the code with this class and its methods are copy/pasted from
other parts of the code, so not much explanation is needed
    database = sqlite3.connect('subnet_db_001.db', timeout=5)
    db = database.cursor()
    db.execute("""CREATE TABLE IF NOT EXISTS "ip_address" (
            "ip" TEXT,
            "subnet" TEXT,
            "network" TEXT,
            "first_host" TEXT,
            "last_host" TEXT,
            "broadcast" TEXT,
            "next_subnet" INTEGER,
```

```
            PRIMARY KEY("ip")
            );""")

    def __init__(self, num1, num2, num3, num4, magic, power, subnet):
        self.num1 = num1
        self.num2 = num2
        self.num3 = num3
        self.num4 = num4
        self.magic = magic
        self.power = power
        self.subnet = subnet

    def answer_class_a(self):
        # print Class A subnets
        self.power = p
        self.magic = m
        network = ipv4_number_format(p * m)
        # For class A, the first host address is always network + 1
        first_host = ipv4_number_format(network + 1)
        # The broadcast address is the magic number plus network address minus
1
        broadcast = ipv4_number_format((network + m) - 1)
        # The last host is broadcast minus 1
        last_host = ipv4_number_format(broadcast - 1)
        # Next subnet needs its own calculator because it can effect every
other octet
        next_subnet = network + m
        # This returns the subnet calculator
        nsa1, nsa2, nsa3, next_subnet = \
            next_sub_calculator(self.num1, self.num2, self.num3, next_subnet)
        # This next command makes returning each answer easier to look at
        ip_address = f"{self.num1}.{self.num2}.{self.num3}.{self.num4}"
        network_address = f"{self.num1}.{self.num2}.{self.num3}.{network}"
        first_host_address = f"{self.num1}.{self.num2}.{self.num3}.{first_host}"
        last_host_address = f"{self.num1}.{self.num2}.{self.num3}.{last_host}"
        broadcast_address = f"{self.num1}.{self.num2}.{self.num3}.{broadcast}"
        next_subnet_address = f"{nsa1}.{nsa2}.{nsa3}.{next_subnet}"
        try:
            # This is meant to insert the answers into the database
            self.db.execute(f"INSERT INTO ip_address "
                            f"VALUES ('{ip_address}', '{self.subnet}',
'{network_address}', "
                            f"'{first_host_address}', '{last_host_address}',
'{broadcast_address}', '{next_subnet_address}')")
            self.database.commit()
            self.database.close()
        except sqlite3.OperationalError:
            """
            There are several of these try statements,
```

```
            the except will not run unless someone runs DROP TABLE ip_address;
in sqlite within the app itself
            the code will break if the table is dropped and the changes are not
saved.
            Until the user saves the database, python will not store the
answers in the database
            """
            print("It seems you may have recently deleted my database without
saving."
                  "\nIf you could save it, that would help a lot")
            pass
        # Returns the class A answers
        return f"Network Address:            {network_address}\n" \
               f"First Host Address:         {first_host_address}\n" \
               f"Last Host Address:          {last_host_address}\n" \
               f"Broadcast Address:          {broadcast_address}\n" \
               f"Next Subnet:                {next_subnet_address}"

    def answer_class_b(self):
        # print Class B subnets
        self.power = p
        self.magic = m
        network = ipv4_number_format(p * m)
        broadcast = (network + m) - 1
        if broadcast >= 255:
            # The third octet doesn't go over 255
            broadcast = 255
        next_subnet = broadcast + 1
        nsa1, nsa2, nsa3, n_subnet_address = \
            next_sub_calculator(self.num1, self.num2, self.num3, next_subnet)
        ip_address = f"{self.num1}.{self.num2}.{self.num3}.{self.num4}"
        network_address = f"{self.num1}.{self.num2}.{network}.{0}"
        first_host_address = f"{self.num1}.{self.num2}.{network}.{1}"
        last_host_address = f"{self.num1}.{self.num2}.{broadcast}.{254}"
        broadcast_address = f"{self.num1}.{self.num2}.{broadcast}.{255}"
        next_subnet_address = f"{nsa1}.{nsa2}.{n_subnet_address}.{0}"
        try:
            self.db.execute(f"INSERT INTO ip_address "
                            f"VALUES ('{ip_address}', '{self.subnet}',
'{network_address}', "
                            f"'{first_host_address}', '{last_host_address}',
'{broadcast_address}', '{next_subnet_address}')")
            self.database.commit()
            self.database.close()
        except sqlite3.OperationalError:
            print("It seems you may have recently deleted my database without
saving."
                  "\nIf you could save it, that would help a lot")
            pass
```

```
        return f"Network Address:            {network_address}\n" \
               f"First Host Address:         {first_host_address}\n" \
               f"Last Host Address:          {last_host_address}\n" \
               f"Broadcast Address:          {broadcast_address}\n" \
               f"Next Subnet:                {next_subnet_address}"

    def answer_class_c(self):
        # prints Class C subnets
        self.power = p
        self.magic = m
        network = ipv4_number_format(p * m)
        broadcast = (network + m) - 1
        next_subnet = broadcast + 1
        nsa1, nsa2, nsa3, n_subnet_address = \
            next_sub_calculator(self.num1, self.num2, self.num3, next_subnet)
        if int(self.num2) >= 255:
            nsa1 += 1
        ip_address = f"{self.num1}.{self.num2}.{self.num3}.{self.num4}"
        network_address = f"{self.num1}.{network}.{0}.{0}"
        first_host_address = f"{self.num1}.{network}.{0}.{1}"
        last_host_address = f"{self.num1}.{broadcast}.{255}.{254}"
        broadcast_address = f"{self.num1}.{broadcast}.{255}.{255}"
        next_subnet_address = f"{nsa1}.{n_subnet_address}.{0}.{0}"
        try:
            self.db.execute(f"INSERT INTO ip_address "
                            f"VALUES ('{ip_address}', '{self.subnet}',
'{network_address}', "
                            f"'{first_host_address}', '{last_host_address}',
'{broadcast_address}', '{next_subnet_address}')")
            self.database.commit()
            self.database.close()
        except sqlite3.OperationalError:
            print("It seems you may have recently deleted my database without
saving."
                  "\nIf you could save it, that would help a lot")
            pass
        return f"Network Address:            {network_address}\n" \
               f"First Host Address:         {first_host_address}\n" \
               f"Last Host Address:          {last_host_address}\n" \
               f"Broadcast Address:          {broadcast_address}\n" \
               f"Next Subnet:                {next_subnet_address}"

    def answer_class_d(self):
        # print Class D subnets
        self.power = p
        self.magic = m
        network = ipv4_number_format(p * m)
        broadcast = (network + m) - 1
        next_subnet = broadcast + 1
```

```
        if self.num1 == 255:
            n_subnet_address = 255
        else:
            nsa1, nsa2, nsa3, n_subnet_address = \
                next_sub_calculator(self.num1, self.num2, self.num3,
next_subnet)
        ip_address = f"{self.num1}.{self.num2}.{self.num3}.{self.num4}"
        network_address = f"{network}.{0}.{0}.{0}"
        first_host_address = f"{network}.{0}.{0}.{1}"
        last_host_address = f"{broadcast}.{255}.{255}.{254}"
        broadcast_address = f"{broadcast}.{255}.{255}.{255}"
        next_subnet_address = f"{n_subnet_address}.{0}.{0}.{0}"
        # This if statement here is calculated just in case the subnet is 0
        if self.subnet == 0:
            next_subnet_address = f"{255}.{255}.{255}.{255}"
        try:
            self.db.execute(f"INSERT INTO ip_address "
                            f"VALUES ('{ip_address}', '{self.subnet}',
'{network_address}', "
                            f"'{first_host_address}', '{last_host_address}',
'{broadcast_address}', '{next_subnet_address}')")
            self.database.commit()
            self.database.close()
        except sqlite3.OperationalError:
            print("It seems you may have recently deleted my database without
saving."
                  "\nIf you could save it, that would help a lot")
            pass
        return f"Network Address:            {network_address}\n" \
               f"First Host Address:         {first_host_address}\n" \
               f"Last Host Address:          {last_host_address}\n" \
               f"Broadcast Address:          {broadcast_address}\n" \
               f"Next Subnet:                {next_subnet_address}"


# The code officially starts here
try:
   # This try/except statement makes sure the user doesn't try to break the
code by entering incorrectly
   # it also makes its own IP address if the user inputs wrong
   ip1, ip2, ip3, ip4 = input("enter an IP address in this format:
'xxx.xxx.xxx.xxx': ").split(".")
   if 255 >= int(ip1) >= 0 and 255 >= int(ip2) >= 0 and 255 >= int(ip3) >= 0
and 255 >= int(ip4) >= 0:
       pass
   else:
       ip1 = random.randint(0, 255)
       ip2 = random.randint(0, 255)
       ip3 = random.randint(0, 255)
```

```
        ip4 = random.randint(0, 255)
        print(f"Here is the IP address I made for you:
{ip1}.{ip2}.{ip3}.{ip4}")

except ValueError:
    # The code fixes itself even if it is broken
    ip1 = random.randint(0, 255)
    ip2 = random.randint(0, 255)
    ip3 = random.randint(0, 255)
    ip4 = random.randint(0, 255)
    print(f"Here is the IP address I made for you: {ip1}.{ip2}.{ip3}.{ip4}")

try:
    # This try/except statement virtually does the same as the one above, but
for the subnet mask
    sub = int(input("Please enter a subnet mask: /"))
    if 32 > int(sub) >= 0:
        pass

    else:
        sub = random.randint(0, 31)
        print(f"Here is the subnet mask I made for you: /{sub}")

except ValueError:
    sub = random.randint(0, 31)
    print(f"Here is the subnet mask I made for you: /{sub}")

database_connect()

# This code checks if the user's ip address was already in the database
answer = database_mine(ip1, ip2, ip3, ip4, sub)

if answer is None:
    # Essentially, if the input isn't in the database, the code continues with
the calculations
    pass
else:
    # This returns the answers straight from the database
    print(f"{answer}\nTurns out I already had the answers in my database!")
    exit()

try:
    # These conditionals are for identifying which class the subnet is
    if 32 > sub >= 24:
        # Class A
        a = ClassA(sub)
        # m is magic number
        m = a.magic()
        # p is used to find the network address
```

```
            p = ensure_magic_number_is_less_than_octet_of_interest(m, ip4)
            answer = PrintAnswer(ip1, ip2, ip3, ip4, m, p, sub)
            print(answer.answer_class_a())

        elif 24 > sub > 16:
            # Class B
            b = ClassB(sub)
            m = b.magic()
            p = ensure_magic_number_is_less_than_octet_of_interest(m, ip3)
            answer = PrintAnswer(ip1, ip2, ip3, ip4, m, p, sub)
            print(answer.answer_class_b())

        elif 16 >= sub >= 8:
            # Class C
            c = ClassC(sub)
            m = c.magic()
            p = ensure_magic_number_is_less_than_octet_of_interest(m, ip2)
            answer = PrintAnswer(ip1, ip2, ip3, ip4, m, p, sub)
            print(answer.answer_class_c())

        elif 8 > sub >= 0:
            # Class D
            d = ClassD(sub)
            m = d.magic()
            p = ensure_magic_number_is_less_than_octet_of_interest(m, ip1)
            answer = PrintAnswer(ip1, ip2, ip3, ip4, m, p, sub)
            print(answer.answer_class_d())

    except sqlite3.IntegrityError:
        """
        This except only passes if the exit() command above on line 477 is disabled
        if the exit() is disabled, the code will attempt to write a copy of the ip
address and answers to the database,
        which it can't because the ip address is the primary key and cannot be
repeated
        """
        pass
```