```
        ############## Application #1 - Part #1 ##############


import random

import sys


def subnet_calc():

    try:

        print "\n"


        #Checking IP address validity

        while True:

            ip_address = raw_input("Enter an IP address: ")


            #Checking octets

            a = ip_address.split('.')


            if (len(a) == 4) and (1 <= int(a[0]) <= 223) and (int(a[0]) != 127) and (int(a[0]) != 169 or int(a[1]) !=
254) and (0 <= int(a[1]) <= 255 and 0 <= int(a[2]) <= 255 and 0 <= int(a[3]) <= 255):

                break


            else:

                print "\nThe IP address is INVALID! Please retry!\n"

                continue


        masks = [255, 254, 252, 248, 240, 224, 192, 128, 0]
```

```
#Checking Subnet Mask validity

while True:

    subnet_mask = raw_input("Enter a subnet mask: ")


    #Checking octets

    b = subnet_mask.split('.')


    if (len(b) == 4) and (int(b[0]) == 255) and (int(b[1]) in masks) and (int(b[2]) in masks) and (int(b[3]) in masks) and (int(b[0]) >= int(b[1]) >= int(b[2]) >= int(b[3])):

        break


    else:

        print "\nThe subnet mask is INVALID! Please retry!\n"

        continue


    ############## Application #1 - Part #2 ##############


#Algorithm for subnet identification, based on IP and Subnet Mask


#Convert mask to binary string

mask_octets_padded = []

mask_octets_decimal = subnet_mask.split(".")

#print mask_octets_decimal


for octet_index in range(0, len(mask_octets_decimal)):
```

```
    #print bin(int(mask_octets_decimal[octet_index]))


    binary_octet = bin(int(mask_octets_decimal[octet_index])).split("b")[1]
    #print binary_octet


    if len(binary_octet) == 8:
        mask_octets_padded.append(binary_octet)


    elif len(binary_octet) < 8:
        binary_octet_padded = binary_octet.zfill(8)
        mask_octets_padded.append(binary_octet_padded)

#print mask_octets_padded


decimal_mask = "".join(mask_octets_padded)
#print decimal_mask   #Example: for 255.255.255.0 => 11111111111111111111111100000000


#Counting host bits in the mask and calculating number of hosts/subnet
no_of_zeros = decimal_mask.count("0")
no_of_ones = 32 - no_of_zeros
no_of_hosts = abs(2 ** no_of_zeros - 2) #return positive value for mask /32


#print no_of_zeros
#print no_of_ones
#print no_of_hosts
```

```
#Obtaining wildcard mask
wildcard_octets = []
for w_octet in mask_octets_decimal:
    wild_octet = 255 - int(w_octet)
    wildcard_octets.append(str(wild_octet))

#print wildcard_octets

wildcard_mask = ".".join(wildcard_octets)
#print wildcard_mask

############## Application #1 - Part #3 ##############

#Convert IP to binary string
ip_octets_padded = []
ip_octets_decimal = ip_address.split(".")

for octet_index in range(0, len(ip_octets_decimal)):

    binary_octet = bin(int(ip_octets_decimal[octet_index])).split("b")[1]

    if len(binary_octet) < 8:
        binary_octet_padded = binary_octet.zfill(8)
        ip_octets_padded.append(binary_octet_padded)
```

```
    else:
        ip_octets_padded.append(binary_octet)

#print ip_octets_padded

binary_ip = "".join(ip_octets_padded)

#print binary_ip   #Example: for 192.168.2.100 => 11000000101010000000001001100100

#Obtain the network address and broadcast address from the binary strings obtained above

network_address_binary = binary_ip[:(no_of_ones)] + "0" * no_of_zeros
#print network_address_binary

broadcast_address_binary = binary_ip[:(no_of_ones)] + "1" * no_of_zeros
#print broadcast_address_binary

net_ip_octets = []
for octet in range(0, len(network_address_binary), 8):
    net_ip_octet = network_address_binary[octet:octet+8]
    net_ip_octets.append(net_ip_octet)

#print net_ip_octets
```

```
net_ip_address = []
for each_octet in net_ip_octets:
    net_ip_address.append(str(int(each_octet, 2)))

#print net_ip_address

network_address = ".".join(net_ip_address)
#print network_address

bst_ip_octets = []
for octet in range(0, len(broadcast_address_binary), 8):
    bst_ip_octet = broadcast_address_binary[octet:octet+8]
    bst_ip_octets.append(bst_ip_octet)

#print bst_ip_octets

bst_ip_address = []
for each_octet in bst_ip_octets:
    bst_ip_address.append(str(int(each_octet, 2)))

#print bst_ip_address

broadcast_address = ".".join(bst_ip_address)
#print broadcast_address
```

```
#Results for selected IP/mask
print "\n"
print "Network address is: %s" % network_address
print "Broadcast address is: %s" % broadcast_address
print "Number of valid hosts per subnet: %s" % no_of_hosts
print "Wildcard mask: %s" % wildcard_mask
print "Mask bits: %s" % no_of_ones
print "\n"

############## Application #1 - Part #4 ##############

#Generation of random IP in subnet
while True:
   generate = raw_input("Generate random ip address from subnet? (y/n)")

   if generate == "y":
      generated_ip = []

      #Obtain available IP address in range, based on the difference between octets in broadcast address and network address
      for indexb, oct_bst in enumerate(bst_ip_address):
         #print indexb, oct_bst
         for indexn, oct_net in enumerate(net_ip_address):
            #print indexn, oct_net
            if indexb == indexn:
               if oct_bst == oct_net:
```

```
                    #Add identical octets to the generated_ip list
                    generated_ip.append(oct_bst)
                else:
                    #Generate random number(s) from within octet intervals and append to the list
                    generated_ip.append(str(random.randint(int(oct_net), int(oct_bst))))

            #IP address generated from the subnet pool
            #print generated_ip
            y_iaddr = ".".join(generated_ip)
            #print y_iaddr

            print "Random IP address is: %s" % y_iaddr
            print "\n"
            continue

        else:
            print "Ok, bye!\n"
            break

    except KeyboardInterrupt:
        print "\n\nProgram aborted by user. Exiting...\n"
        sys.exit()

#Calling the function
subnet_calc()
```