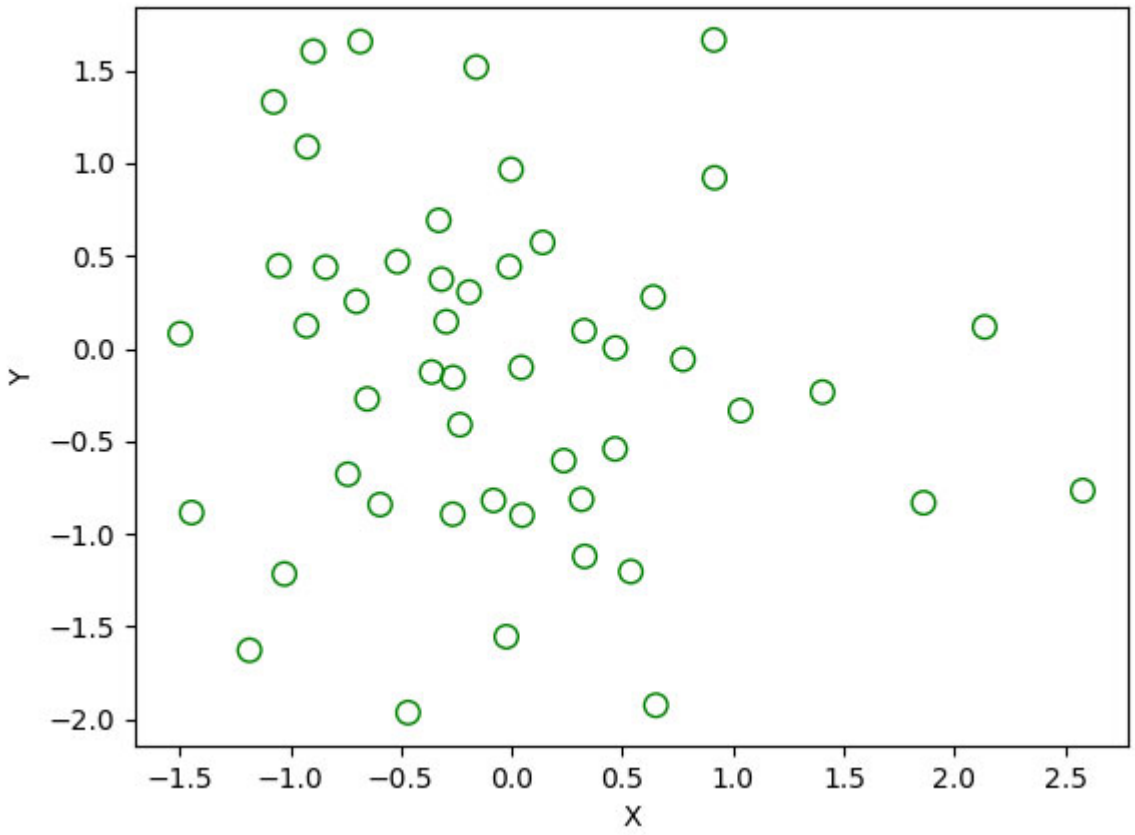
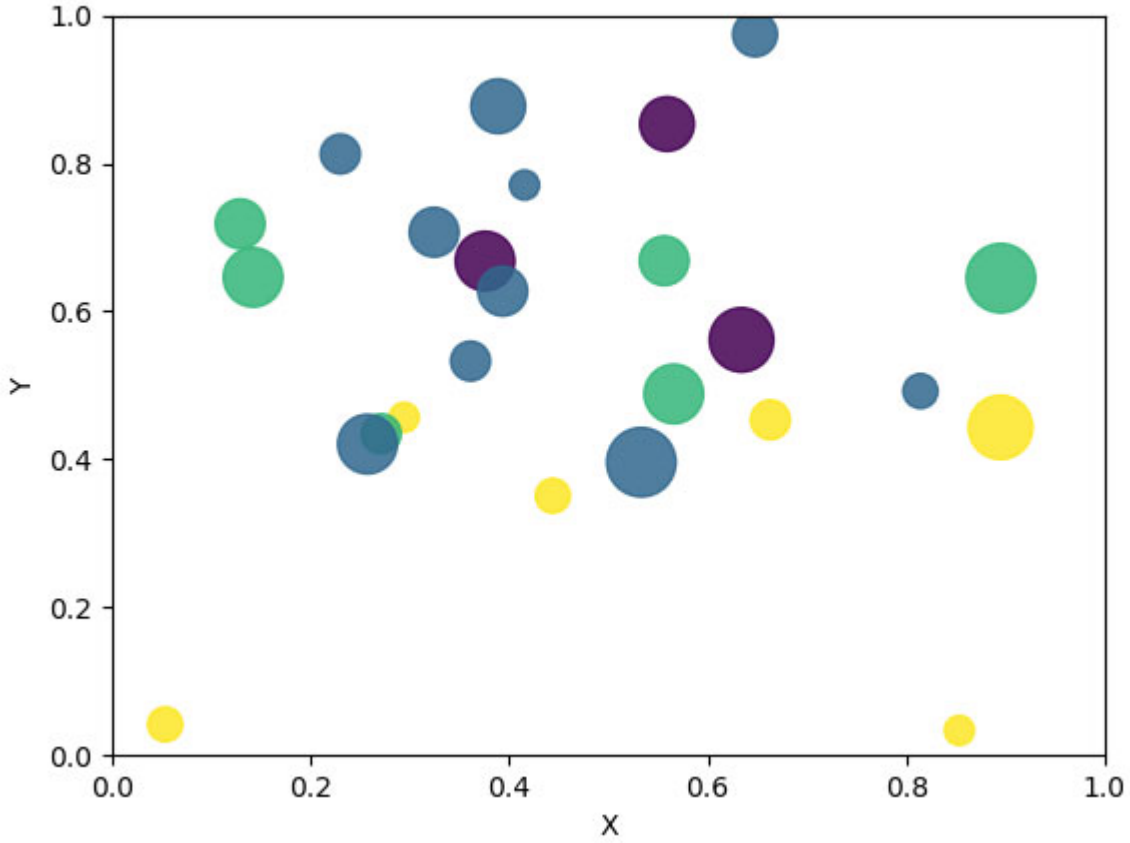
```
# Q1
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(50)
y = np.random.randn(50)
plt.scatter(x, y, s=70, facecolors='none', edgecolors='g')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

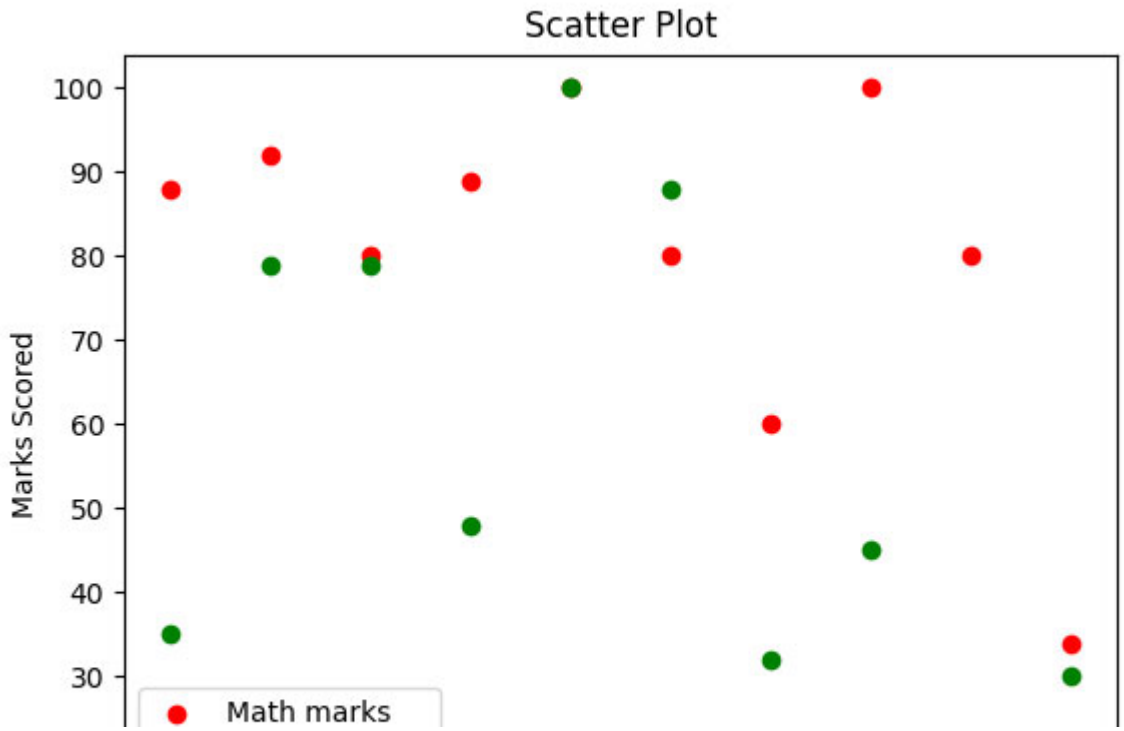



```
# Q2
import math
import random
import matplotlib.pyplot as plt
# create random data
no_of_balls = 25
x = [random.triangular() for i in range(no_of_balls)]
y = [random.gauss(0.5, 0.25) for i in range(no_of_balls)]
colors = [random.randint(1, 4) for i in range(no_of_balls)]
areas = [math.pi * random.randint(5, 15)**2 for i in range(no_of_balls)]
# draw the plot
plt.figure()
plt.scatter(x, y, s=areas, c=colors, alpha=0.85)
```

```
plt.axis([0.0, 1.0, 0.0, 1.0])
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```




```
# Q3
import matplotlib.pyplot as plt
import pandas as pd
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(marks_range, math_marks, label='Math marks', color='r')
plt.scatter(marks_range, science_marks, label='Science marks', color='g')
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()
```




```
# Q4
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
dic4 = {}
for d in (dic1, dic2, dic3): dic4.update(d)
print(dic4)
```

```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

```
# Q5
d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
def is_key_present(x):
  if x in d:
      print('Key is present in the dictionary')
  else:
      print('Key is not present in the dictionary')

is_key_present(3)
```

```
Key is present in the dictionary
```

```
# Q6
n = 6
d = dict()
for i in range(1, n+1):
    d[i] = i*i
print(d)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

```
# Q7
mydict = {1  : 2, 3 : 4, 5 : 6}
mydict.pop(3)
print(mydict)
```

```
{1: 2, 5: 6}
```

```
# Q8
keys = ['black', 'white', 'red']
values = ['#000000','#ffffff', '#ff0000']
color_dictionary = dict(zip(keys, values))
print(color_dictionary)
```

```
{'black': '#000000', 'white': '#ffffff', 'red': '#ff0000'}
```

```
# Q9

dict1 = {'a': 1, 'b' : 2, 'c' : 1, 'd' : 4, 'e' : 2, 'g' : 5, 'e' : 4, 'f' : 5}
mark = list()

for i in range(len(dict1)):
    mark.append(False)

u_list = list()

for i, x in zip(dict1.keys(), range(len(dict1))):
    if(mark[x] == True):
        continue
    u_list.append(dict1[i])
    mark[x] = True

    for j, y in zip(dict1.keys(), range(len(dict1))):
        if(dict1[i] == dict1[j]):
            mark[y] = True;

print(u_list)
```

```
[1, 2, 4, 5]
```

```
# Q10
str1 = "hellohappiestWorld"
dict1 = {}

mark = list()

for i in range(len(str1)):
```

```
        mark.append(False)

    for i in range(len(str1)):
        if mark[i] == True :
            continue
        curr_count = 1

        for j in range(i + 1, len(str1)):
            if(str1[i] == str1[j]):
                curr_count += 1
                mark[j] = True

        dict1[str1[i]] = curr_count

    print(dict1)
```

```
     {'h': 2, 'a': 2, 'p': 2, 'y': 2, 'o': 1, 'l': 1, 'i': 1, 'd': 1, 's': 1}
```

```
    # Q11
    string = "I am the greatest ever"
    arr = string.split();
    string = "-".join(arr);
    print(string)
```

```
     I-am-the-greatest-ever
```

```
    # Q12
    import random
    randList = []
    for i in range(10):
        n = random.randint(100,200)
        randList.append(n)
    print(min(randList))
```

```
     106
```

```
    # Q13
    countryDict = {'India': 'Rupee', 'Japan': 'Yen', 'USA': 'Dollar', 'Switzerland': 'Francs', 'U
    for i in countryDict:
        print(i)
```

```
     India
     Japan
     USA
     Switzerland
     UAE
```

```
    # Q14
    cmplx1 = complex(2, 3)
```

```
cmplx2 = 3 + 3j;
print(cmplx1.imag)
print(cmplx1.real)
conj = cmplx1.conjugate();
abs1 = abs(cmplx1)
print(abs1)
```

```
<class 'complex'>
2.0
3.0
(1.9999999999999996-3j)
```

```
# Q15
str1 = "  hello   "
str2 = str1.strip()[0:2] + "lp"
print(str2)
```

```
Help
```

```
# Q16
# Same as Q12
```

```
# Q17
str1 = "motherfucker"
last_index = len(str1) - 1
str2 = str1[last_index] + str1[1 : last_index] + str1[0]
print(str2)
```

```
ywertQ
```

```
# Q18
string = "The sun shines bright everyday"
substring = "hines"
if (substring in string):
    print("This substring exists")
else:
    print("The substring doesn't exist")
```

```
 This substring exists
```

```
# Q19
# Same as Q11
```

```
# Q20
str1 = "maam"
len1 = int((len(str1) + 1)/2)
```

```
flag = True

for x, y in zip(range(len1), range(len(str1) - 1, len(str1) - len1 - 1, -1)):
    print(x, y)
    if str1[x] != str1[y]:
        flag = False
        break

if flag == False:
    print("Not a palindrome")
else :
    print("A palindrome")
```

```
 It is a palidrome
```

```
# Q21
# string = "life is unfair at times"
# print(string.title())

str1 = "hi i am parth"
divide = str1.split()
new_str = str
for i in range(len(divide)):
    curr = divide[i]
    divide[i] = curr[0].upper() + curr[1:len(curr)]

string1 = " ".join(divide)
print(string1)
```

```
 Life Is Unfair At Times
```

```
# Q22
string = "mathematics is hard as heck"
x = "h"
counter = 0
for ch in string:
    if x == ch:
        counter += 1
print(f"The number of times {x} occurs in the string is {counter}")
```

```
 The number of times h occurs in the string is 3
```

```
# Q23
myDict = {"India" : "New Delhi", "England" : "London", "Phillipines" : "Manilla"}
myDict["France"] = "Paris"
print(myDict)
```

```
{'India': 'New Delhi', 'England': 'London', 'Phillipines': 'Manilla', 'France': 'Paris'}
```

```
# Q24

def countCharacterType(str):

    vowels = 0
    consonant = 0
    specialChar = 0
    digit = 0

    for i in range(0, len(str)):

        ch = str[i]

        if ( (ch >= 'a' and ch <= 'z') or (ch >= 'A' and ch <= 'Z') ):

            ch = ch.lower()

            if (ch == 'a' or ch == 'e' or ch == 'i'or ch == 'o' or ch == 'u'):
                vowels += 1
            else:
                consonant += 1

        elif (ch >= '0' and ch <= '9'):
            digit += 1
        else:
            specialChar += 1

    print("Vowels:", vowels)
    print("Consonant:", consonant)
    print("Digit:", digit)
    print("Special Character:", specialChar)


str = "life is a highway54"
countCharacterType(str)
```

```
Vowels: 6
Consonant: 8
Digit: 2
Special Character: 3
```

```
# Q25
string = "life is like an ice cream"
for x in range(len(string)):
    print(f"{x} : {string[x]}")
```

```
0 : l
1 : i
2 : f
3 : e
4 :
```

```
5 : i
6 : s
7 :
8 : l
9 : i
10 : k
11 : e
12 :
13 : a
14 : n
15 :
16 : i
17 : c
18 : e
19 :
20 : c
21 : r
22 : e
23 : a
24 : m
```

```
# Q26
import numpy as np
a = np.ones(3, dtype = int)
print("Matrix a :", a)
```

```
Matrix a : [1 1 1]
```

```
# Q27
arr = np.array([[1, 2, 3, 4, 5],
                  [6, 7, 8, 9, 10],
                  [11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20]
                  ])
print([1, 2, 3, 4, 5] in arr.tolist())
print([16, 17, 20, 19, 18] in arr.tolist())
```

```
True
False
```

```
#28
import numpy as np

arr1 = np.array([[1, 2, 3], [4, 5, 6], [1, 2, 3]])
arr2 = np.array([[4, 5, 6], [1, 2, 3], [4, 5, 6]])
print(arr1 + arr2)
print(arr1 - arr2)
print(arr1 * arr2)
print(arr1.dot(arr2))
print(arr1 @ arr2)
```

```
[[5 7 9]
 [5 7 9]
 [5 7 9]]
[[-3 -3 -3]
 [ 3  3  3]
 [-3 -3 -3]]
[[ 4 10 18]
 [ 4 10 18]
 [ 4 10 18]]
[[18 24 30]
 [45 60 75]
 [18 24 30]]
[[18 24 30]
 [45 60 75]
 [18 24 30]]
```

```
# Q29
import numpy as np

x = np.array([1,2,3,4,5,1,2,1,1,1])
uniq, count = np.unique(x, return_counts = True)
print(uniq)
print(count)

# print(max(count))
# print(min(count), uniq.tolist()[count.tolist().index(min(count))])
# print(max(count), uniq.tolist()[count.tolist().index(max(count))])
max_count = max(count)
index1 = count.tolist().index(max_count)
print(uniq[index1])
```

```
[1 2 3 4 5]
[5 2 1 1 1]
1
```

```
# Q30
ini_array = np.array([[1, 2, 3], [2, 4, 5], [1, 2, 3]])

print("Initial array:\n", str(ini_array))

result = ini_array.flatten()

print("New resulting array:", result)
```

```
Initial array:
 [[1 2 3]
 [2 4 5]
 [1 2 3]]
New resulting array: [1 2 3 2 4 5 1 2 3]
```

```
# Q31
```

```
TwoDList = [[1, 2, 3], [4, 5, 6],
            [7, 8, 9], [10, 11, 12]]
TwoDArray = np.array(TwoDList)

print("2D Array:")
print(TwoDArray)

print("\nColumn-wise Sum:", np.sum(TwoDArray, axis = 0))
```

```
2D Array:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

Column-wise Sum: [22 26 30]
78
```

```
#### Practice ####
arr2 = np.array([[1,2,3], [4,5,6]])
print(arr2[1][0])
```

```
4
```

```
# Q32
list = [2, 4, 4, 4, 5, 5, 7, 9]

print(np.var(list), np.average(list), np.std(list))
```

```
4.0 5.0 2.0
```

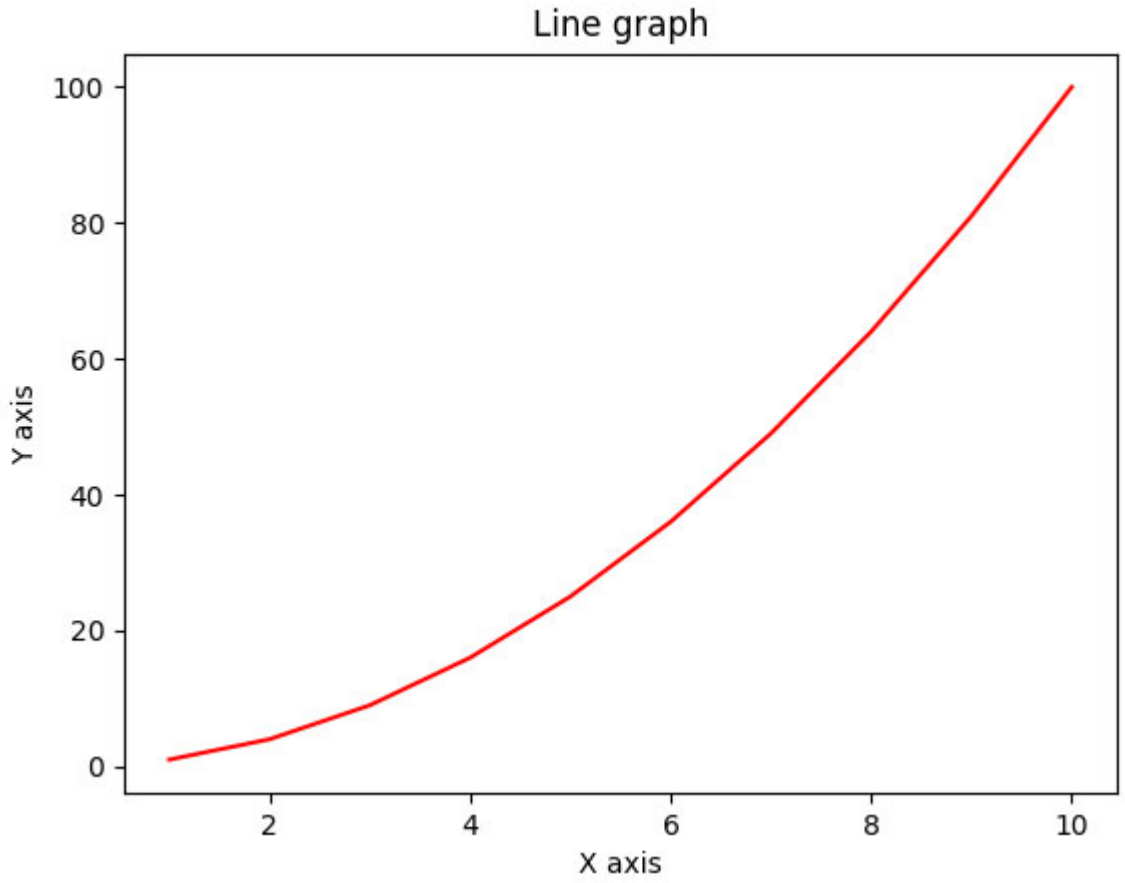
```
# Q33
import numpy as np
x = np.array(['python exercises', 'PHP', 'java', 'C++'], dtype=str)
print("Original Array:")
print(x)
r = np.char.join(" ", x)
print(r)
```

```
Original Array:
['python exercises' 'PHP' 'java' 'C++']
['p y t h o n   e x e r c i s e s' 'P H P' 'j a v a' 'C + +']
```

```
# Q34
x = np.arange(1, 11)
y = x * x

plt.title("Line graph")
plt.xlabel("X axis")
plt.ylabel("Y axis")
```

```
plt.plot(x, y, color ="red")
plt.show()
```




```
# Q35
Number = 123456
Reverse = 0
while(Number > 0):
    Reminder = Number %10
    Reverse = (Reverse *10) + Reminder
    Number = Number //10

print("\n Reverse of entered number is = %d" %Reverse)
```

```
 Reverse of entered number is = 654321
```

```
# Q36
n = 7
curr_sum = 1

for i in range (1, n + 1):
    print(" 1", end = "")
```

```
    for j in range (2, i + 1):
        curr_sum += j
        print(" +",j, end = "")

    print(" =", curr_sum)
```

```
  1 = 1
  1 + 2 = 3
  1 + 2 + 3 = 8
  1 + 2 + 3 + 4 = 17
  1 + 2 + 3 + 4 + 5 = 31
  1 + 2 + 3 + 4 + 5 + 6 = 51
  1 + 2 + 3 + 4 + 5 + 6 + 7 = 78
```

```
# Q37
# limit = 40
# c = 0
# m = 2
# print(f"All the possible triplets with an upper limit of {limit} are:")
# while(c<limit):
#     for n in range(1,m+1):
#         a = m * m - n * n
#         b = 2 * m * n
#         c = m * m + n * n
#         if(c > limit):
#             break
#         if(a == 0 or b == 0 or c == 0):
#             break
#         print(a, b, c)
#     m = m + 1

import math

n = 2000
i = 1
sqrt_n = int(math.sqrt(n))
for i in range(2, sqrt_n):
    for j in range(i + 1, sqrt_n):
        x = i * i + j * j
        rootx = math.sqrt(x)
        isSquare = ((rootx * 10) % 10) == 0
        if(x <= n and isSquare):
            print(i, j, int(rootx))
```

```
 All the possible triplets with an upper limit of 40 are:
 3 4 5
 8 6 10
 5 12 13
 15 8 17
 12 16 20
 7 24 25
 24 10 26
```

```
 21 20 29
 16 30 34
```

```
# Q38
num1 = 111111010
str1 = str(num1)
diff1 = abs(str1.count("1") - str1.count("0"))
if(diff1 >= len(str1) - 2):
    print("Yes possible")
else :
    print("Not possible")
```

```
#Q.39
nums = [8, 12, 7, 4, 11, 6, 3, 2, 5, 13]
great_inx = -1
steps = 0

n = len(nums)
for i in range(n):
    great = -100
    for j in range(n):
        if(nums[j] > great):
            great = nums[j]
            great_inx = j

    if great_inx - n != 0:
        steps += 1
        nums.pop(great_inx)
    n -= 1

print(steps)
```

```
 10
```

```
# Q40
n = 6

for i in range(1, n + 1):
    for j in range(1, i + 1):
        print(f" {i} ", end = "")
    print("")
```

```
  1
  2  2
  3  3  3
  4  4  4  4
  5  5  5  5  5
  6  6  6  6  6  6
```

```
|# Q41
keys = ["Rash", "Kil", "Varsha"]
values = [1, 4, 5]

res = dict(zip(keys, values))

print ("Resultant dictionary is : " +  str(res))
```

```
Resultant dictionary is : {'Rash': 1, 'Kil': 4, 'Varsha': 5}
```

```
# Q42
num1 = 2324
count = 0

while(num1 > 10):
    count += 1
    num1 = num1 // 10

print(count)
```

```
3
```

```
# Q43
fileObject = open("/sample.txt", "r")
data = fileObject.read()
output = data.title()
print(output)
```

```
Hey This Is A Sample File
```

```
# Q44
fileObject = open("sample.txt", "r")
data = fileObject.read()
print(data.count("a"))
```

```
2
```

```
# Q45
file1 = open("sample.txt", "a")
file1.write("\n")
file1.write("Life is a lie")
fileObject = open("sample.txt", "r")
data = fileObject.read()
print(data)
```

```
This is a sample txt file
Life is a lie
```

```
# Q46
fileObject = open("sample.txt", "r")
data = fileObject.read()
print(data)
```

```
     This is a sample txt file
     Life is a lie
```

```
# Q47
fileObject = open("sample.txt", "r")
data = fileObject.read()
fileObject2 = open("newSample.txt", "w")
fileObject2.writelines(data)
fileObject2.close()
```

```
# Q48
name = input("Enter the name of the student: ")
rollno = int(input("Enter the rollno of the student: "))
fileObject3 = open("newFile3.txt", "w")
fileObject3.writelines("Name: "+name+"\n"+"Roll No: "+str(rollno))
fileObject3.close()
```

```
     Enter the name of the student: Yash Bijoor
     Enter the rollno of the student: 60003200136
```

```
# Q49
fileName = input("Enter the path of the file: ")
fileObject = open("sample.txt", "r")
data = fileObject.read()
newfileObject = open(fileName+".txt", "w")
newfileObject.writelines(data)
newfileObject.close()
```

```
     Enter the path of the file: sample2
```

```
#Q.50

file = open("/sample.txt", 'r')

for line in file :
    for word in line.split():
        print(word)
```

```
     hey
     this
     is
     a
     sample
     file
```

```
and
I
love
a
sample
file
```

```
# Q51
file = open("/sample.txt", 'r')
for lines in file:
  for words in lines.split():
    for chars in words:
      print(chars)
```

```
h
e
y
t
h
i
s
i
s
a
s
a
m
p
l
e
f
i
l
e
a
n
d
I
l
o
v
e
a
s
a
m
p
l
e
f
i
l
e
```

```
# Q52
file2 = open("/sample.txt", 'r')
total_lines = 0
total_words = 0
total_spaces = 0
total_chars = 0
for line in file2:
    total_lines += 1
    total_words += len(line.split())
    total_spaces += len(line.split()) - 1
    for word in line.split():
        total_chars += len(word)

print("The total lines : " + str(total_lines) + " words : " + str(total_words) + " spaces : "
```

```
# Q53
marks = 105
class Error(Exception):
  """Base class for other exceptions"""
  pass
class InvalidMarks(Error):
  pass
try:
  print("The marks entered is:", marks)
  if marks > 100:
    raise InvalidMarks
except InvalidMarks:
  print("Marks entered is greater than 100")
```

```
    The marks entered is: 105
    Marks entered is greater than 100
```

```
# Q54
a, b, c, d = 2, 3, 4, 0
class Error(Exception):
  """Base class for other exceptions"""
  pass
class UserDefinedError(Error):
  pass

try:
  if b*d == 0:
    raise UserDefinedError
  else:
    print("The required value is ", ((a+d) + (b*c))/ (b*d))
except UserDefinedError:
  print("The value of (b*d) is 0. Hence, invalid.")
```

```
     The value of (b*d) is 0. Hence, invalid.
```

```
# Q55
age = 17
class Error(Exception):
  """Base class for other exceptions"""
  pass
class AgeError(Error):
  pass

try:
  if age < 18:
    raise AgeError
  else:
    print("Age is valid")
except AgeError:
  print("Age is not valid")
```

```
     Age is not valid
```

```
# Q56
try:
  fileObject = open("randomNonExistentFile.txt", "r")
except FileNotFoundError:
  print("File not found!!")
```

```
     File not found!!
```

```
# Q57
num = 57
try:
  print("The entered number is " + num)
except TypeError:
  print("A float type can't be concatenated with a string")
```

```
     A float type can't be concatenated with a string
```

```
# Q58
class Complex:
  def __init__(self, real, imag):
    self.real = real
    self.imag = abs(imag)
  def add(self):
    return self.real + self.imag
num1 = Complex(4, 7j)
addition = num1.add()
print(addition)
```

```
11.0
```

```
# Q59
class Triangle:
  def __init__(self, sides):
    self.sides = sides

  def get_perimeter(self):
    return self.sides[0] + self.sides[1] + self.sides[2]

triangle1 = Triangle([7, 8, 9])
perimeter = triangle1.get_perimeter()
print("The perimeter of the triangle is", perimeter)
```

```
The perimeter of the triangle is 24
```

```
# Q60
class List:
  def __init__(self, lst):
    self.lst = lst

  def append(self, x):
    self.lst.append(x)

  def delete(self, x):
    self.lst.remove(x)

  def print_list(self):
    print(self.lst)

list1 = List([1, 2, 3, 4, 5])
list1.append(6)
list1.delete(3)
list1.print_list()
```

```
[1, 2, 4, 5, 6]
```

```
# Q61
class Calc:
  def __init__(self, n1, n2):
    self.n1 = n1
    self.n2 = n2

  def add(self):
    return self.n1 + self.n2

  def subtract(self):
```

```
    return self.n1 - self.n2

  def multiply(self):
    return self.n1 * self.n2

  def divide(self):
    return self.n1 / self.n2

calc1 = Calc(8, 3)
print(calc1.add(), calc1.subtract(), calc1.multiply(), calc1.divide())
```

```
 11 5 24 2.6666666666666665
```

```
# Q62
class Student:
    def __init__(self,name, id):
        self.id = id
        self.name = name

    def printer(self):
        try :
            print("Name :", self.name, "\nId :", self.id, "\nClass :", self.clss)
        except Exception as e:
            print("\nName :", self.name, "\nId :", self.id)


stud1 = Student("parthVaghela", 116)
stud1.clss = "B division"
stud1.printer()
```

```
 Name : parthVaghela
 Id : 116
 Class : B division
```

```
# Q63
string = "Life is but a dream"
arr = string.split()[::-1]      #[start, stop, step(if negative then start stepping from stop
revString = " ".join(arr)
print(revString)
```

```
 dream a but is Life
```

```
# Q64
class String:
  def get_string(self):
    self.string = input("Enter a string: ")

  def print_string(self):
    print(self.string.upper())
```

```
string1 = String()
string1.get_string()
string1.print_string()
```

```
     Enter a string: fuck off guys
     FUCK OFF GUYS
```

```
# Q65
import math
class Circle:
  def __init__(self, radius):
    self.radius = radius

  def area(self):
    return math.pi*self.radius*self.radius

  def perimeter(self):
    return 2*math.pi*self.radius

circle1 = Circle(7)
print("The area of the circle is %.2f"%circle1.area())
print("The perimeter of the circle is %.2f"%circle1.perimeter())
```

```
     The area of the circle is 153.94
     The perimeter of the circle is 43.98
```

```
# Q66
class Vehicle:
  def __init__(self, max_speed, mileage):
    self.max_speed = max_speed
    self.mileage = mileage

  def print_attributes(self):
    print(self.max_speed, self.mileage)

class Bus(Vehicle):
  pass

bus = Bus(180, 92)
bus.print_attributes()
```

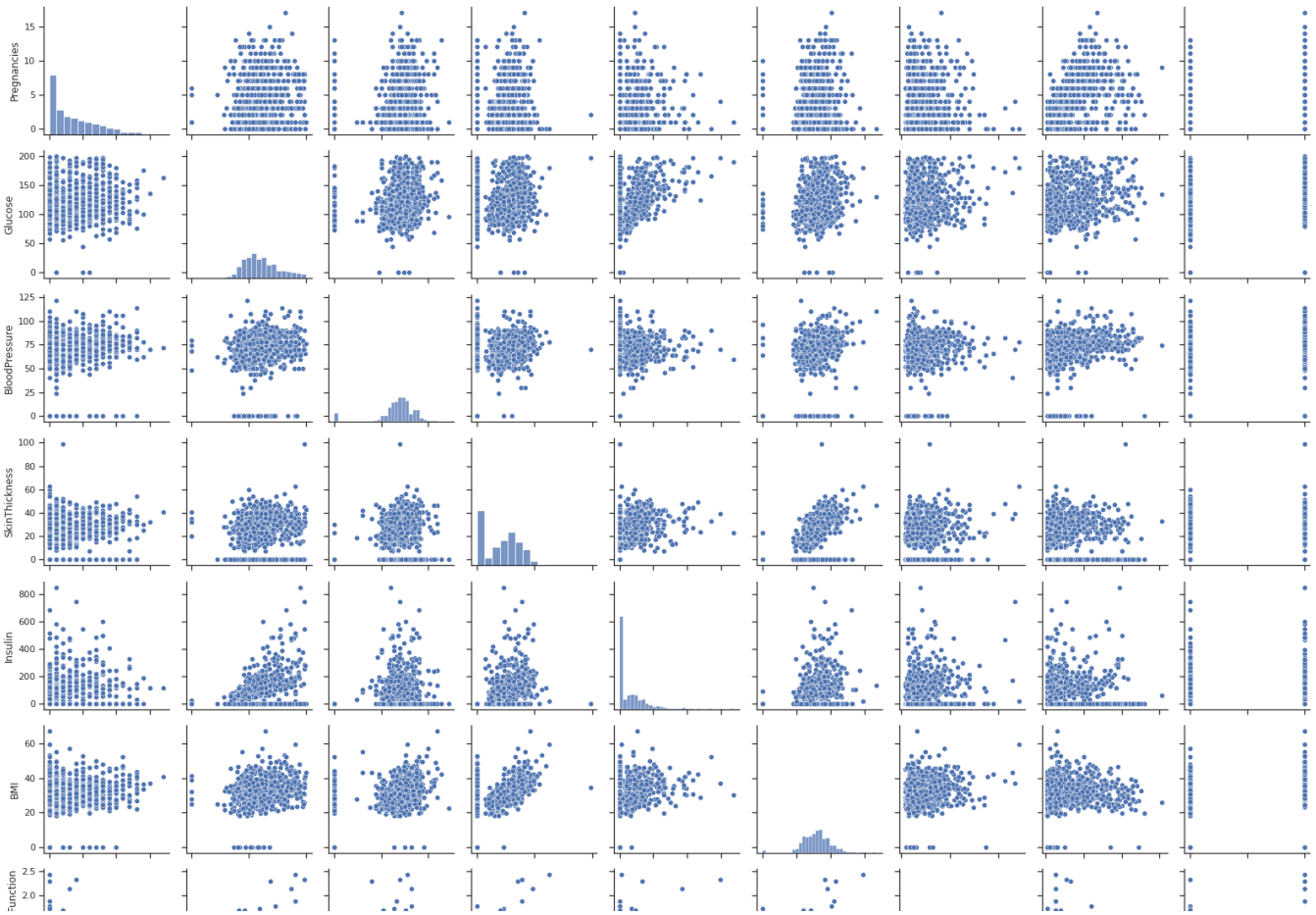
```
     180 92
```

```
# Q67
import pandas as pd
da = pd.read_csv("/diabetes.csv")
```

```
da.describe()
```

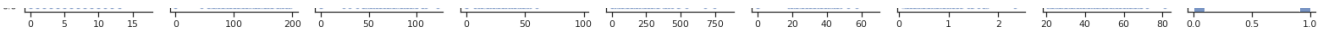
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style='ticks',color_codes=True)
sns.pairplot(da)
plt.show()
```




```
# Q68
lst = [1, 2, 0, 0, 8, 9, 0, 7, 0, 1]
n = lst.count(0)
for i in range(n):
  lst.remove(0)
  lst.append(0)
print(lst)
```

```
[1, 2, 8, 9, 7, 1, 0, 0, 0, 0]
```



```
# Q69
str1 = "hello,how,are,you,doing"
list1 = str1.split(",")

list2 = sorted(list1)
str3 = ",".join(list2)
print(str3)
```

```
are,doing,hello,how,you
```

```
# Q70
import math
inp = input("Enter the value of C, D and H separated by a comma: ")
values = [value for value in inp.split(",")]
C, D, H = int(values[0]), int(values[1]), int(values[2])
```

```
Q = math.sqrt((2*C*D)/H)
print(Q)
```

```
Enter the value of C, D and H separated by a comma: 2,4,1
4.0
```

```
# Q71
list1 = [12, 24, 35, 24, 88, 120, 155, 88, 120, 155]
n = len(list1)

i = 0
while(i < n):
    j = i + 1
    while(j < n):
        if list1[i] == list1[j]:
            list1.pop(j)
            j -= 1
            n -= 1
        j += 1
    i += 1
print(list1)
```

```
[12, 24, 35, 88, 120, 155]
```

```
def printDict(n):
    dict1 = dict()
    for i in range(1, n + 1):
        dict1[i] = i * i
    print(dict1)

printDict(8)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}
```

```
# Q73
num1 = 353483
sum1 = 0
while(num1 > 0):
    sum1 += num1%10
    num1 //= 10
print(sum1)
```

```
26
```

```
# Q74
n = 100
list1 = list(x for x in range(0, n + 1))
```

```
for x in range(2,n + 1):
    if(list1[x] != -1):
        print(x," ", end = "")
        temp = x
        i = 1
        while(temp <= n):
            list1[temp] = -1
            i += 1
            temp = x * i
    else:
        continue
```

```
 2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73  79  83
```

```
# Q75
def dig(n):
  if n>0:
    print(n%10, end=" ")
    dig(n//10)
n = 3245654
dig(n)
```

```
 4 5 6 5 4 2 3
```

```
# Q76
def studentData(std_id, **var):
    print("student id : ", std_id)
    if('std_name' and 'std_class' in var):
        print("student name :", var['std_name'], "\nstudent class :", var['std_class'])

    elif('std_name' in var):
        print("student name :", var['std_name'])

studentData(45)
studentData(35, std_name = 'hello')
studentData(25, std_name = 'parth', std_class = 'B' )
```

```
 Student ID: SV12
 Student Name: Jean Garner
```

```
# Q77
class Solution(object):
   def romanToInt(self, s):
      roman = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000,'IV':4,'IX':9,'XL':40,'XC':
      i = 0
      num = 0
```

```
        while i < len(s):
           if i+1<len(s) and s[i:i+2] in roman:
              num+=roman[s[i:i+2]]
              i+=2
           else:
              #print(i)
              num+=roman[s[i]]
              i+=1
        return num
ob1 = Solution()
print(ob1.romanToInt("XCVII"))
```

```
97
```

```
# Q78
class Solution:
    def __init__(self,list1):
        self.list1 = list1

    def subs(self):
        self.list1 = list1
        len1 = len(self.list1)
        for length in range(len1 + 1):
            i = 0
            j = i + length
            while (j < len1):
                print(self.list1[i:j + 1])
                i += 1
                j += 1
list1 = [1, 2, 3, 4, 5]
ob1 = Solution(list1)
ob1.subs()
```

```
[1]
[2]
[3]
[4]
[5]
[1, 2]
[2, 3]
[3, 4]
[4, 5]
[1, 2, 3]
[2, 3, 4]
[3, 4, 5]
[1, 2, 3, 4]
[2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

```
# Q79
```

```
list1 = [10, 7, 3, 2, 5, 4, 8]
n = len(list1)
list1 = sorted(list1)
print(list1)
given_sum = 9
i, j = 0, n - 1

while(i != j):
    cursum = list1[i] + list1[j]
    if cursum > given_sum:
        j -= 1
    elif cursum < given_sum:
        i += 1
    else:
        print(list1[i], list1[j])
        i += 1
```

```
 [2, 3, 4, 5, 7, 8, 10]
 2 7
 4 5
```

```
# Q80 Write a Python class to implement pow(x, n).
class Power:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def power(self):
        power1 = self.a
        for i in range(self.b - 1):
            power1 *= self.a
        return power1

ob1 = Power(2, 3)
print(ob1.power())
```

```
 8
```

```
# Q81
import numpy as np
fvalues = [0, 12, 45.21, 34, 99.91, 32]
F = np.array(fvalues)
print("Values in Fahrenheit degrees:")
print(F)
print("Values in  Centigrade degrees:")
print(np.round((5*F/9 - 5*32/9),2))
```

```
 Values in Fahrenheit degrees:
 [ 0.   12.   45.21 34.   99.91 32.  ]
```

```
Values in  Centigrade degrees:
[-17.78 -11.11   7.34   1.11  37.73   0.  ]
```

```
# Q82
array1 = np.array([0, 10, 20, 40, 60, 80])
print("Array1:",array1)
array2 = [10, 30, 40, 50, 70]
print("Array2:",array2)
print("Unique values in array1 that are not in array2:", np.setdiff1d(array1, array2))
```

```
Array1: [ 0 10 20 40 60 80]
Array2: [10, 30, 40, 50, 70]
Unique values in array1 that are not in array2: [ 0 20 60 80]
```

```
# Q83
import pandas as pd
exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthe
        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam_data, index=labels)
print(df)
```

```
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

```
# Q84
print("First three rows of the data frame:")
print(df.iloc[:3])
```

```
First three rows of the data frame:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
```

```
# Q85
print("Select specific columns:")
print(df[['name', 'score']])
```

```
     Select specific columns:
             name  score
     a  Anastasia   12.5
     b       Dima    9.0
     c  Katherine   16.5
     d      James    NaN
     e      Emily    9.0
     f    Michael   20.0
     g    Matthew   14.5
     h      Laura    NaN
     i      Kevin    8.0
     j      Jonas   19.0
```

```
# Q86
print("Select specific columns and rows:")
print(df.iloc[[1, 3, 5, 6], [1, 3]])
```

```
     Select specific columns and rows:
        score qualify
     b    9.0      no
     d    NaN      no
     f   20.0     yes
     g   14.5     yes
```

```
# Q87
print("Number of attempts in the examination is greater than 2:")
print(df[df['attempts'] > 2])
```

```
     Number of attempts in the examination is greater than 2:
           name  score  attempts qualify
     b     Dima    9.0         3      no
     d    James    NaN         3      no
     f  Michael   20.0         3     yes
```

```
# Q88
print("Number of Rows:", len(df.axes[0]))
print("Number of Columns: ", len(df.axes[1]))
```

```
     Number of Rows: 10
     Number of Columns:  4
```

```
# Q89
print(df[df['score'].between(15, 20)])
```

```
             name  score  attempts qualify
     c  Katherine   16.5         2     yes
     f    Michael   20.0         3     yes
     j      Jonas   19.0         1     yes
```

```
# Q90
```

```
print(df[(df['attempts'] < 2) & (df['score'] > 15)])  #ampersand is important***
```

```
     name  score  attempts qualify
j  Jonas   19.0         1     yes
```

```
# Q91
df.loc['k'] = [1, 'Suresh', 'yes', 15.5]
print("After inserting a new record:")
print(df)
print("\nAfter deleting the new row:")
df = df.drop('k')   #see here the substitution
print(df)
```

```
After inserting a new record:
        name   score attempts qualify
a  Anastasia    12.5        1     yes
b       Dima     9.0        3      no
c  Katherine    16.5        2     yes
d      James     NaN        3      no
e      Emily     9.0        2      no
f    Michael    20.0        3     yes
g    Matthew    14.5        1     yes
h      Laura     NaN        1      no
i      Kevin     8.0        2      no
j      Jonas    19.0        1     yes
k          1  Suresh      yes    15.5

After deleting the new row:
        name score attempts qualify
a  Anastasia  12.5        1     yes
b       Dima   9.0        3      no
c  Katherine  16.5        2     yes
d      James   NaN        3      no
e      Emily   9.0        2      no
f    Michael  20.0        3     yes
g    Matthew  14.5        1     yes
h      Laura   NaN        1      no
i      Kevin   8.0        2      no
j      Jonas  19.0        1     yes
```

```
# Q92
df.sort_values(by=['name', 'score'], ascending=[False, True])
print(df)
```

```
        name score attempts qualify
a  Anastasia  12.5        1     yes
b       Dima   9.0        3      no
c  Katherine  16.5        2     yes
d      James   NaN        3      no
e      Emily   9.0        2      no
f    Michael  20.0        3     yes
g    Matthew  14.5        1     yes
h      Laura   NaN        1      no
```

```
      i      Kevin   8.0        2      no
      j      Jonas  19.0        1     yes
```

```
# Q93
df['qualify'] = df['qualify'].map({'yes': True, 'no': False})
print(df)
```

```
            name score attempts  qualify
     a  Anastasia  12.5        1     True
     b       Dima   9.0        3    False
     c  Katherine  16.5        2     True
     d      James   NaN        3    False
     e      Emily   9.0        2    False
     f    Michael  20.0        3     True
     g    Matthew  14.5        1     True
     h      Laura   NaN        1    False
     i      Kevin   8.0        2    False
     j      Jonas  19.0        1     True
```

```
# Q94
df.at[2,'score']=10
print(df)
```

```
            name  score  attempts qualify
     0  Anastasia   12.5         1     yes
     1       Dima    9.0         3      no
     2  Katherine   10.0         2     yes
     3      James    NaN         3      no
     4      Emily    9.0         2      no
     5    Michael   20.0         3     yes
     6    Matthew   14.5         1     yes
     7      Laura    NaN         1      no
     8      Kevin    8.0         2      no
     9      Jonas   19.0         1     yes
```