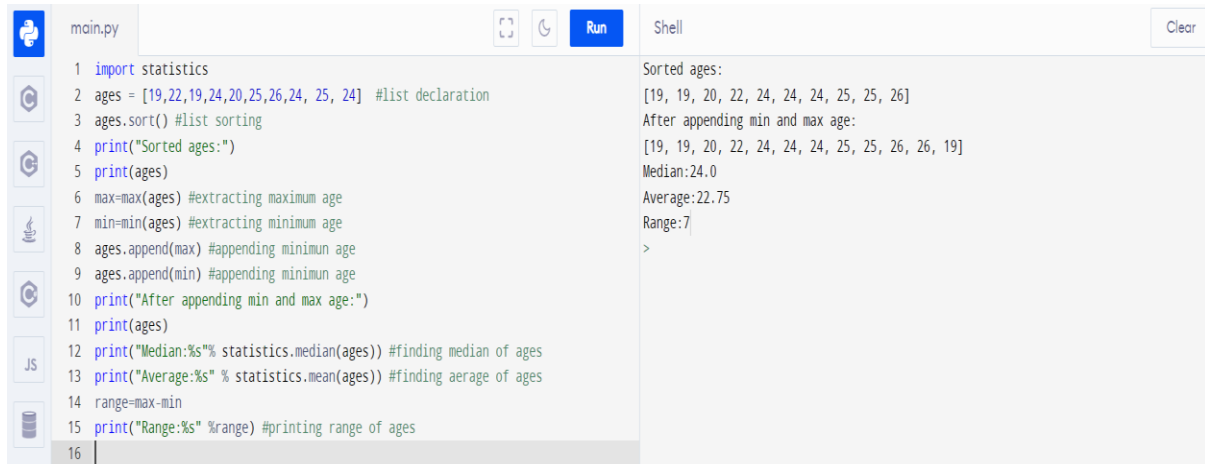


1. The following is a list of 10 students ages: ages = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24]

- Sort the list and find the min and max age
- Add the min age and the max age again to the list
- Find the median age (one middle item or two middle items divided by two)
- Find the average age (sum of all items divided by their number)
- Find the range of the ages (max minus min)

Screenshot:



```
main.py
1 import statistics
2 ages = [19,22,19,24,20,25,26,24, 25, 24] #list declaration
3 ages.sort() #list sorting
4 print("Sorted ages:")
5 print(ages)
6 max=max(ages) #extracting maximum age
7 min=min(ages) #extracting minimum age
8 ages.append(max) #appending minimum age
9 ages.append(min) #appending minimum age
10 print("After appending min and max age:")
11 print(ages)
12 print("Median:%s" % statistics.median(ages)) #finding median of ages
13 print("Average:%s" % statistics.mean(ages)) #finding aaverage of ages
14 range=max-min
15 print("Range:%s" %range) #printing range of ages
16
```

Shell

```
Sorted ages:
[19, 19, 20, 22, 24, 24, 24, 25, 25, 26]
After appending min and max age:
[19, 19, 20, 22, 24, 24, 24, 25, 25, 26, 26, 19]
Median:24.0
Average:22.75
Range:7
>
```

Initialized ages list and then sorted ages using sort() function and then assigned variables for min, max ages and then added them to the original list and then calculated median, average and range using inbuilt functions.

Source Code:

1. import statistics
2. ages = [19,22,19,24,20,25,26,24, 25, 24] #list declaration
3. ages.sort() #list sorting
4. print("Sorted ages:")
5. print(ages)
6. max=max(ages) #extracting maximum age
7. min=min(ages) #extracting minimum age
8. ages.append(max) #appending minimum age
9. ages.append(min) #appending minimum age
10. print("After appending min and max age:")
11. print(ages)
12. print("Median:%s" % statistics.median(ages)) #finding median of ages
13. print("Average:%s" % statistics.mean(ages)) #finding aaverage of ages
14. range=max-min
15. print("Range:%s" %range) #printing range of ages

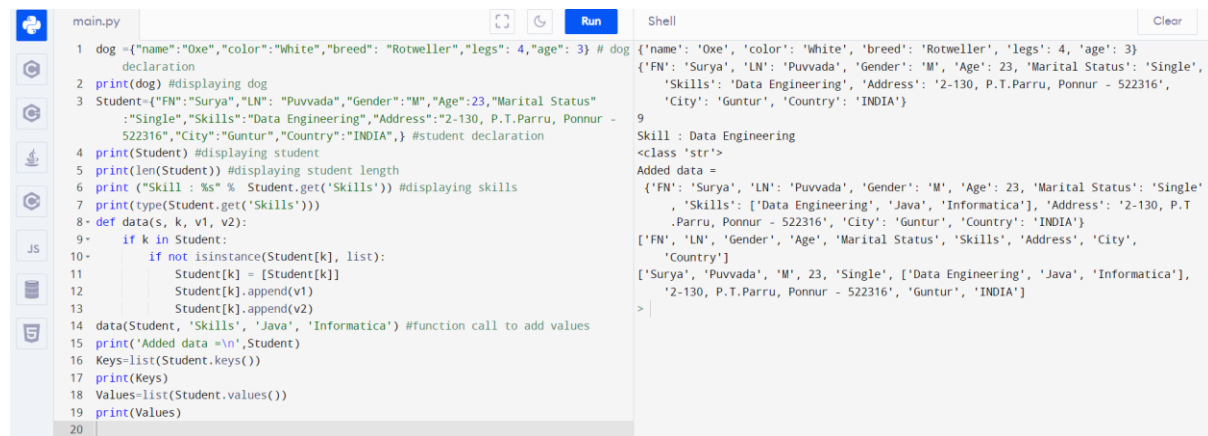
Output:

```
Sorted ages:
[19, 19, 20, 22, 24, 24, 24, 25, 25, 26]
After appending min and max age:
[19, 19, 20, 22, 24, 24, 24, 25, 25, 26, 26, 19]
Median:24.0
Average:22.75
Range:7
>
```

2.

- Create an empty dictionary called dog
- Add name, color, breed, legs, age to the dog dictionary
- Create a student dictionary and add first_name, last_name, gender, age, marital status, skills, country, city and address as keys for the dictionary
- Get the length of the student dictionary
- Get the value of skills and check the data type, it should be a list
- Modify the skills values by adding one or two skills
- Get the dictionary keys as a list
- Get the dictionary values as a list

Screenshot:



The screenshot shows a code editor with a file named 'main.py' and a shell window. The code in 'main.py' defines a 'dog' dictionary, a 'Student' dictionary, and a 'data' function. The 'Student' dictionary has keys for 'FN', 'LN', 'Gender', 'Age', 'Marital Status', 'Skills', 'Address', 'City', and 'Country'. The 'data' function appends new skills to the 'Skills' list. The shell window shows the output of the code, including the 'dog' dictionary, the 'Student' dictionary, the length of the 'Student' dictionary, the 'Skills' list, and the keys and values of the 'Student' dictionary.

```
1 dog ={"name":"Oxe","color":"White","breed": "Rotweiler","legs": 4,"age": 3} # dog
  declaration
2 print(dog) #displaying dog
3 Student={"FN":"Surya","LN": "Puvvada","Gender":"M","Age":23,"Marital Status"
  : "Single","Skills":"Data Engineering","Address":"2-130, P.T.Parru, Ponnur -
  522316","City":"Guntur","Country":"INDIA",} #student declaration
4 print(Student) #displaying student
5 print(len(Student)) #displaying student length
6 print ("Skill : %s" % Student.get('Skills')) #displaying skills
7 print(type(Student.get('Skills')))
8 def data(s, k, v1, v2):
9     if k in Student:
10         if not isinstance(Student[k], list):
11             Student[k] = [Student[k]]
12             Student[k].append(v1)
13             Student[k].append(v2)
14 data(Student, 'Skills', 'Java', 'Informatica') #function call to add values
15 print('Added data =\n',Student)
16 Keys=list(Student.keys())
17 print(Keys)
18 Values=list(Student.values())
19 print(Values)
20
```

```
{'name': 'Oxe', 'color': 'White', 'breed': 'Rotweiler', 'legs': 4, 'age': 3}
{'FN': 'Surya', 'LN': 'Puvvada', 'Gender': 'M', 'Age': 23, 'Marital Status': 'Single',
 'Skills': 'Data Engineering', 'Address': '2-130, P.T.Parru, Ponnur - 522316',
 'City': 'Guntur', 'Country': 'INDIA'}
9
Skill : Data Engineering
<class 'str'>
Added data =
{'FN': 'Surya', 'LN': 'Puvvada', 'Gender': 'M', 'Age': 23, 'Marital Status': 'Single',
 'Skills': ['Data Engineering', 'Java', 'Informatica'], 'Address': '2-130, P.T
 .Parru, Ponnur - 522316', 'City': 'Guntur', 'Country': 'INDIA'}
['FN', 'LN', 'Gender', 'Age', 'Marital Status', 'Skills', 'Address', 'City',
 'Country']
['Surya', 'Puvvada', 'M', 23, 'Single', ['Data Engineering', 'Java', 'Informatica'],
 '2-130, P.T.Parru, Ponnur - 522316', 'Guntur', 'INDIA']
>
```

Declared given dog dictionary and displayed the dog. Declared student dictionary and then added the required values to the student and the length is calculated using len function and for getting skills declared data function which takes 4 values and then calling data function to append the skills and finally keys and values of set are retrieved using keys() and values() functions by calling with student object.

Source Code:

1. dog={"name":"Oxe","color":"White","breed": "Rotweiler","legs": 4,"age": 3} # dog declaration
2. print(dog) #displaying dog
3. Student={"FN":"Surya","LN":"Puvvada","Gender":"M","Age":23,"Marital Status":"Single","Skills":"Data Engineering","Address":"2-130, P.T.Parru, Ponnur - 522316","City":"Guntur","Country":"INDIA",} #student declaration
4. print(Student) #displaying student
5. print(len(Student)) #displaying student length
6. print ("Skill : %s" % Student.get('Skills')) #displaying skills
7. print(type(Student.get('Skills')))
8. def data(s, k, v1, v2):
9. if k in Student:
10. if not isinstance(Student[k], list):
11. Student[k] = [Student[k]]
12. Student[k].append(v1)
13. Student[k].append(v2)
14. data(Student, 'Skills', 'Java', 'Informatica') #function call to add values
15. print('Added data =\n',Student)
16. Keys=list(Student.keys())
17. print(Keys)
18. Values=list(Student.values())
19. print(Values)

Output:

```
{'name': 'Oxe', 'color': 'White', 'breed': 'Rotweiler', 'legs': 4, 'age': 3}
{'FN': 'Surya', 'LN': 'Puvvada', 'Gender': 'M', 'Age': 23, 'Marital Status': 'Single', 'Skills': 'Data Engineering',
'Address': '2-130, P.T.Parru, Ponnur - 522316', 'City': 'Guntur', 'Country': 'INDIA'}
9
Skill : Data Engineering
<class 'str'>
Added data =
{'FN': 'Surya', 'LN': 'Puvvada', 'Gender': 'M', 'Age': 23, 'Marital Status': 'Single', 'Skills': ['Data Engineering', 'Java',
'Informatica'], 'Address': '2-130, P.T.Parru, Ponnur - 522316', 'City': 'Guntur', 'Country': 'INDIA'}
['FN', 'LN', 'Gender', 'Age', 'Marital Status', 'Skills', 'Address', 'City', 'Country']
['Surya', 'Puvvada', 'M', 23, 'Single', ['Data Engineering', 'Java', 'Informatica'], '2-130, P.T.Parru, Ponnur - 522316',
'Guntur', 'INDIA']
>
```

3.

- Create a tuple containing names of your sisters and your brothers (imaginary siblings are fine)
- Join brothers and sisters tuples and assign it to siblings
- How many siblings do you have?
- Modify the siblings tuple and add the name of your father and mother and assign it to family_members

Screenshot:



The screenshot shows a Python IDE with a file named 'main.py'. The code in the editor is as follows:

```
1 brothers=("Ganesh","Kumar","Seetha","Kesav","Masthan") #Brother Tuple Creation
2 print(brothers)
3 sisters=("Supriya","Nirmala","Razia","Mahitha") #Sister Tuple Creation
4 print(sisters)
5 siblings=brothers+sisters #Merging brother and sister tuples
6 print(siblings)
7 print("Siblings Count:",len(siblings)) #printing tuple length
8 siblings=("Srinivas","Padmavathi")+siblings #adding father and mother
9 print("Siblings:",siblings) #printing after parents added
10 family_members=siblings #assigning siblings to family members
11 print("Family Members:",family_members) #printing family members
12
```

The output in the Shell window is:

```
('Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan')
('Supriya', 'Nirmala', 'Razia', 'Mahitha')
('Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan', 'Supriya', 'Nirmala', 'Razia',
'Mahitha')
Siblings Count: 9
Siblings: ('Srinivas', 'Padmavathi', 'Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan',
'Supriya', 'Nirmala', 'Razia', 'Mahitha')
Family Members: ('Srinivas', 'Padmavathi', 'Ganesh', 'Kumar', 'Seetha', 'Kesav',
'Masthan', 'Supriya', 'Nirmala', 'Razia', 'Mahitha')
```

Created two tuples separately for brothers and sisters and added values to those and added brothers with sisters which results in the list of siblings. The number of siblings can be retrieved using length function applying for the sibling's tuple. Added father and mother name to the sibling tuple and assigned it to the family members and printed them.

Source Code:

1. brothers=("Ganesh","Kumar","Seetha","Kesav","Masthan") #Brother Tuple Creation
2. print(brothers)
3. sisters=("Supriya","Nirmala","Razia","Mahitha") #Sister Tuple Creation
4. print(sisters)
5. siblings=brothers+sisters #Merging brother and sister tuples
6. print(siblings)
7. print("Siblings Count:",len(siblings)) #printing tuple length
8. siblings=("Srinivas","Padmavathi")+siblings #adding father and mother
9. print("Siblings:",siblings) #printing after parents added
10. family_members=siblings #assigning siblings to family members
11. print("Family Members:",family_members) #printing family members

Output:

('Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan')

('Supriya', 'Nirmala', 'Razia', 'Mahitha')

('Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan', 'Supriya', 'Nirmala', 'Razia', 'Mahitha')

Siblings Count: 9

Siblings: ('Srinivas', 'Padmavathi', 'Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan', 'Supriya', 'Nirmala', 'Razia', 'Mahitha')

Family Members: ('Srinivas', 'Padmavathi', 'Ganesh', 'Kumar', 'Seetha', 'Kesav', 'Masthan', 'Supriya', 'Nirmala', 'Razia', 'Mahitha')

>

4.

```
it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}
```

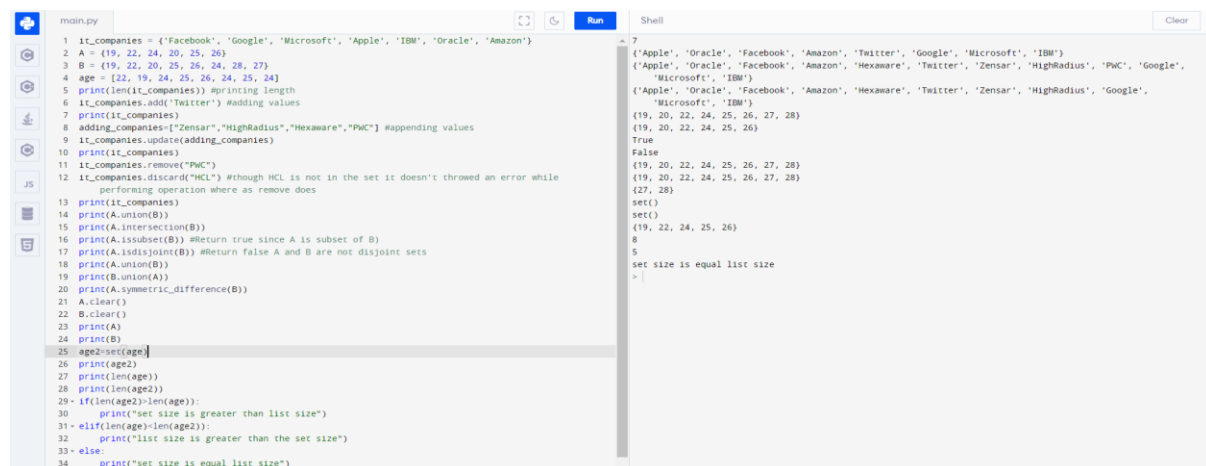
```
A = {19, 22, 24, 20, 25, 26}
```

```
B = {19, 22, 20, 25, 26, 24, 28, 27}
```

```
age = [22, 19, 24, 25, 26, 24, 25, 24]
```

- Find the length of the set `it_companies`
- Add 'Twitter' to `it_companies`
- Insert multiple IT companies at once to the set `it_companies`
- Remove one of the companies from the set `it_companies`
- What is the difference between `remove` and `discard`
- Join A and B
- Find A intersection B
- Is A subset of B
- Are A and B disjoint sets
- Join A with B and B with A
- What is the symmetric difference between A and B
- Delete the sets completely
- Convert the ages to a set and compare the length of the list and the set.

Screenshot:



```
main.py
1 it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}
2 A = {19, 22, 24, 20, 25, 26}
3 B = {19, 22, 20, 25, 26, 24, 28, 27}
4 age = [22, 19, 24, 25, 26, 24, 25, 24]
5 print(len(it_companies)) #printing length
6 it_companies.add('Twitter') #adding values
7 print(it_companies)
8 adding_companies = {"Zensar", "HighRadius", "Hexaware", "PwC"} #appending values
9 it_companies.update(adding_companies)
10 print(it_companies)
11 it_companies.remove("PwC")
12 it_companies.discard("HCL") #though HCL is not in the set it doesn't throw an error while
   performing operation where as remove does
13 print(it_companies)
14 print(A.union(B))
15 print(A.intersection(B))
16 print(A.issubset(B)) #Return true since A is subset of B
17 print(A.isdisjoint(B)) #Return false A and B are not disjoint sets
18 print(A.union(B))
19 print(B.union(A))
20 print(A.symmetric_difference(B))
21 A.clear()
22 B.clear()
23 print(A)
24 print(B)
25 age2 = set(age)
26 print(age2)
27 print(len(age))
28 print(len(age2))
29 if (len(age2) > len(age)):
30     print("set size is greater than list size")
31 elif (len(age) > len(age2)):
32     print("list size is greater than the set size")
33 else:
34     print("set size is equal list size")

Shell
7
8 {'Apple', 'Oracle', 'Facebook', 'Amazon', 'Twitter', 'Google', 'Microsoft', 'IBM'}
9 {'Apple', 'Oracle', 'Facebook', 'Amazon', 'Hexaware', 'Twitter', 'Zensar', 'HighRadius', 'PwC', 'Google',
   'Microsoft', 'IBM'}
10 {'Apple', 'Oracle', 'Facebook', 'Amazon', 'Hexaware', 'Twitter', 'Zensar', 'HighRadius', 'Google',
   'Microsoft', 'IBM'}
11 {19, 20, 22, 24, 25, 26, 27, 28}
12 {19, 20, 22, 24, 25, 26}
13 True
14 False
15 {19, 20, 22, 24, 25, 26, 27, 28}
16 {19, 20, 22, 24, 25, 26, 27, 28}
17 {27, 28}
18 set()
19 {19, 22, 24, 25, 26}
20 8
21 8
22 set size is equal list size
23 >
```

Taken input for `it_companies`, A, B, age and the length of the companies is taken by using `length` function. Tried to add companies to variable and then tried to delete companies use `remove` function and then all the given union, intersection, subset, disjoint conditions are implemented using python inbuilt functions. Then age is assigned age2 as a set values and compared the length of the age and age2 and displaying output accordingly.

Source Code:

1. `it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}`
2. `A = {19, 22, 24, 20, 25, 26}`
3. `B = {19, 22, 20, 25, 26, 24, 28, 27}`
4. `age = [22, 19, 24, 25, 26, 24, 25, 24]`

```

5. print(len(it_companies)) #printing length
6. it_companies.add("Twitter") #adding values
7. print(it_companies)
8. adding_companies=["Zensar","HighRadius","Hexaware","PWC"] #appending values
9. it_companies.update(adding_companies)
10. print(it_companies)
11. it_companies.remove("PWC")
12. it_companies.discard("HCL") #though HCL is not in the set it doesn't throw an error while performing
    operation where as remove does
13. print(it_companies)
14. print(A.union(B))
15. print(A.intersection(B))
16. print(A.issubset(B)) #Return true since A is subset of B
17. print(A.isdisjoint(B)) #Return false A and B are not disjoint sets
18. print(A.union(B))
19. print(B.union(A))
20. print(A.symmetric_difference(B))
21. A.clear()
22. B.clear()
23. print(A)
24. print(B)
25. age2=set(age)
26. print(age2)
27. print(len(age))
28. print(len(age2))
29. if(len(age2)>len(age)):
30. print("set size is greater than list size")
31. elif(len(age)<len(age2)):
32. print("list size is greater than the set size")
33. else:
34. print("set size is equal list size")

```

Output:

```

7
{'Apple', 'Oracle', 'Facebook', 'Amazon', 'Twitter', 'Google', 'Microsoft', 'IBM'}
{'Apple', 'Oracle', 'Facebook', 'Amazon', 'Hexaware', 'Twitter', 'Zensar', 'HighRadius', 'PWC', 'Google', 'Microsoft',
'IBM'}
{'Apple', 'Oracle', 'Facebook', 'Amazon', 'Hexaware', 'Twitter', 'Zensar', 'HighRadius', 'Google', 'Microsoft', 'IBM'}
{19, 20, 22, 24, 25, 26, 27, 28}
{19, 20, 22, 24, 25, 26}
True
False
{19, 20, 22, 24, 25, 26, 27, 28}
{19, 20, 22, 24, 25, 26, 27, 28}
{27, 28}
set()
set()
{19, 22, 24, 25, 26}
8
5
set size is equal list size
>

```

5. The radius of a circle is 30 meters.

- Calculate the area of a circle and assign the value to a variable name of `_area_of_circle_`
- Calculate the circumference of a circle and assign the value to a variable name of `_circum_of_circle_`
- Take radius as user input and calculate the area.

Screenshot:



```
main.py
1 radius=30
2 print("Default Radius:%s" % radius) #printing given radius
3- def area(t): #function declaration for area
4     area_of_circle=3.14*(t)**2 #formula
5     print("Area: %s mt/s^2" % area_of_circle)
6- def circum(t): #function declaration for circumference
7     circum_of_circle=2*3.14*(t) #formula
8     print("Circumference: %s mt/s" % circum_of_circle)
9 area(radius) #function call
10 circum(radius)
11- while(True):
12     radius=float(input("Enter radius:"))
13     if(radius==0):
14         break
15     elif(radius>0):
16         area(radius) #calling area function
17         circum(radius) #calling circumference function
18     elif(radius<0):
19         print("Radius can be calculated for positive values")
20     else:
21         print("Please enter a valid input")

Shell
Default Radius:30
Area: 2826.0 mt/s^2
Circumference: 188.4 mt/s
Enter radius:6
Area: 113.04 mt/s^2
Circumference: 37.68 mt/s
Enter radius:-2
Radius can be calculated for positive values
Enter radius:2.354
Area: 17.399732240000002 mt/s^2
Circumference: 14.783120000000002 mt/s
Enter radius:0
>
```

Taken given radius input and then defined a function for area and calculated using the area of circle formula and similarly calculating circumference. Then taking radius input from the user and if the input entered is valid, calculated the area and circumference otherwise it will ask for valid input.

Source Code:

```
1. radius=30
2. print("Default Radius:%s" % radius) #printing given radius
3. def area(t): #function declaration for area
4.     area_of_circle=3.14*(t)**2 #formula
5.     print("Area: %s mt/s^2" % area_of_circle)
6. def circum(t): #function declaration for circumference
7.     circum_of_circle=2*3.14*(t) #formula
8.     print("Circumference: %s mt/s" % circum_of_circle)
9. area(radius) #function call
10. circum(radius)
11. while(True):
12.     radius=float(input("Enter radius:"))
13.     if(radius==0):
14.         break
15.     elif(radius>0):
16.         area(radius) #calling area function
17.         circum(radius) #calling circumference function
18.     elif(radius<0):
19.         print("Radius can be calculated for positive values")
20.     else:
21.         print("Please enter a valid input")
```

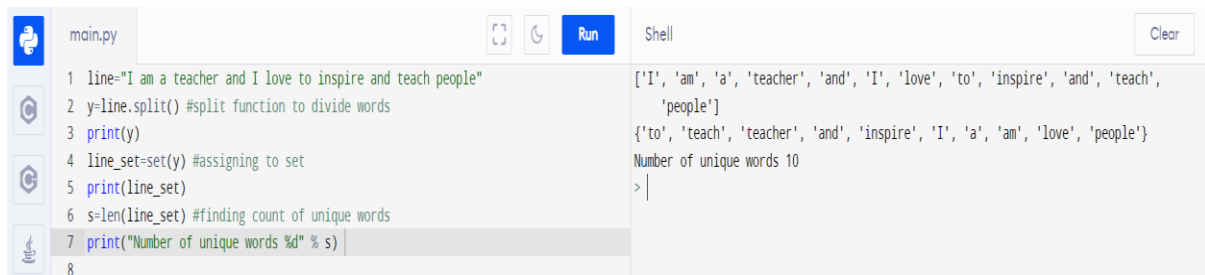
Output:

```
Default Radius:30
Area: 2826.0 mt/s^2
Circumference: 188.4 mt/s
Enter radius:6
Area: 113.04 mt/s^2
Circumference: 37.68 mt/s
Enter radius:-2
Radius can be calculated for positive values
Enter radius:2.354
Area: 17.399732240000002 mt/s^2
Circumference: 14.783120000000002 mt/s
Enter radius:0
>
```

6. “I am a teacher and I love to inspire and teach people”

- How many unique words have been used in the sentence? Use the split methods and set to get the unique words.

Screenshot:



```
main.py
1 line="I am a teacher and I love to inspire and teach people"
2 y=line.split() #split function to divide words
3 print(y)
4 line_set=set(y) #assigning to set
5 print(line_set)
6 s=len(line_set) #finding count of unique words
7 print("Number of unique words %d" % s)
8
```

```
Shell
['I', 'am', 'a', 'teacher', 'and', 'I', 'love', 'to', 'inspire', 'and', 'teach', 'people']
{'to', 'teach', 'teacher', 'and', 'inspire', 'I', 'a', 'am', 'love', 'people'}
Number of unique words 10
>
```

Taken line variable to take the input and then declared a variable y which will take the individual words using a split function for line. Then applying length function for y will give you the number of individual unique words used in the string.

Source Code:

1. line="I am a teacher and I love to inspire and teach people"
2. y=line.split() #split function to divide words
3. print(y)
4. line_set=set(y) #assigning to set
5. print(line_set)
6. s=len(line_set) #finding count of unique words
7. print("Number of unique words %d" % s)

Output:

```
['I', 'am', 'a', 'teacher', 'and', 'I', 'love', 'to', 'inspire', 'and', 'teach', 'people']
{'to', 'teach', 'teacher', 'and', 'inspire', 'I', 'a', 'am', 'love', 'people'}
Number of unique words 10
>
```

7. Use a tab escape sequence to get the following lines.

```
Name      Age  Country  City
Asabeneh 250  Finland  Helsinki
```

Screenshot:



```
main.py
1 print("Name\tAge\tCountry\t City") #printing header
2 print("Asabeneh\t250\tFinland\t Helsinki") #printing values
```

```
Shell
Name      Age  Country  City
Asabeneh 250  Finland  Helsinki
>
```

‘\t’ the tab key is used to display the given format. Each tab is used for one space and for the printed format wherever required used two for a decent look.

Source Code:

1. print("Name\tAge\tCountry\t City") #printing header
2. print("Asabeneh\t250\tFinland\t Helsinki") #printing values

Output:

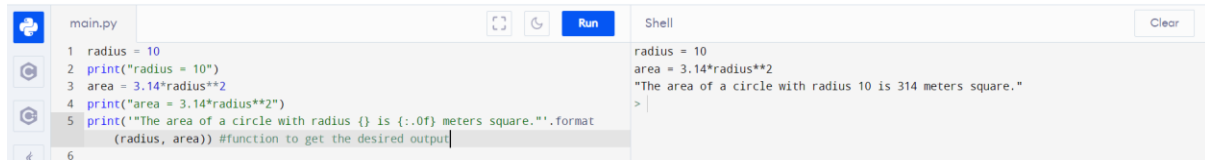
```
Name      Age  Country  City
Asabeneh 250  Finland  Helsinki
>
```

8. Use the string formatting method to display the following:

```
radius = 10
area = 3.14 * radius ** 2
```

“The area of a circle with radius 10 is 314 meters square.”

Screenshot:



```
main.py
1 radius = 10
2 print("radius = 10")
3 area = 3.14*radius**2
4 print("area = 3.14*radius**2")
5 print("The area of a circle with radius {} is {:.0f} meters square.".format
  (radius, area)) #function to get the desired output
6
```

```
Shell
radius = 10
area = 3.14*radius**2
"The area of a circle with radius 10 is 314 meters square."
>
```

Taken inputs for the radius and area variable declared with the formula and then for the given format used print function to display the output.

Source Code:

1. radius = 10
2. print("radius = 10")
3. area = 3.14*radius**2
4. print("area = 3.14*radius**2")
5. print("The area of a circle with radius {} is {:.0f} meters square.".format(radius, area)) #function to get the desired output

Output:

```
radius = 10
area = 3.14*radius**2
"The area of a circle with radius 10 is 314 meters square."
>
```

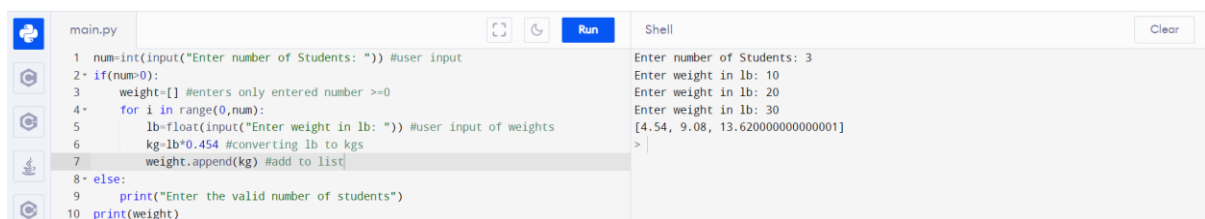
9. Write a program, which reads weights (lbs.) of N students into a list and convert these weights to kilograms in a separate list using Loop.

N: No of students (Read input from user)

Ex: L1: [150, 155, 145, 148]

Output: [68.03, 70.3, 65.77, 67.13]

Screenshot:



```
main.py
1 num=int(input("Enter number of Students: ")) #user input
2 if(num>0):
3     weight=[] #enters only entered number >=0
4     for i in range(0,num):
5         lb=float(input("Enter weight in lb: ")) #user input of weights
6         kg=lb*0.454 #converting lb to kgs
7         weight.append(kg) #add to list
8 else:
9     print("Enter the valid number of students")
10 print(weight)
```

```
Shell
Enter number of Students: 3
Enter weight in lb: 10
Enter weight in lb: 20
Enter weight in lb: 30
[4.54, 9.08, 13.620000000000001]
>
```

Num variable is taken to get the number of students and then if the entered number is a valid one then it will asks for the weights for the individual students and using the lb to kg conversion formula it will calculate the KG and added that to weight and finally displayed weight in KG's.

Source Code:

1. num=int(input("Enter number of Students: ")) #user input
2. if(num>0):
3. weight=[] #enters only entered number >=0
4. for i in range(0,num):
5. lb=float(input("Enter weight in lb: ")) #user input of weights
6. kg=lb*0.454 #converting lb to kgs
7. weight.append(kg) #add to list
8. else:

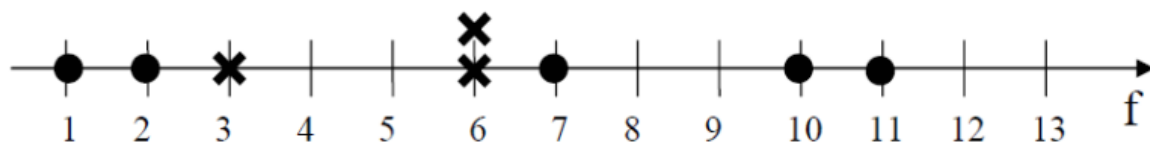

```
9. print("Enter the valid number of students")
10. print(weight)
```

Output:

```
Enter number of Students: 3
Enter weight in lb: 10
Enter weight in lb: 20
Enter weight in lb: 30
[4.54, 9.08, 13.620000000000001]
>
```

10.

The diagram below shows a dataset with 2 classes and 8 data points, each with only one feature value, labeled f. Note that there are two data points with the same feature value of 6. These are shown as two x's one above the other.



1. Divide this data equally into two parts. Use first part as training and second part as testing. Using KNN classifier, for K=3, what would be the predicted outputs for the test samples? Show how you arrived at your answer.
2. Compute the confusion matrix for this and calculate accuracy, sensitivity and specificity values.

Screenshot:

```
import numpy as nm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
X = nm.array([[1, 0], [2, 0], [3, 0], [6, 0], [6, 0], [7, 0], [10, 0], [11, 0]])
print("Dataset: \n", X, "\n Size: ", len(X))
y = nm.array([0, 0, 1, 1, 1, 0, 0, 0])
print("\n Class label for the datasets X Y: ", y)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.50,
                                                    random_state=0, shuffle=False)
print("\n Training Data X: \n", x_train)
print("\n Training Data label Y: ", y_train)
# feature Scaling-used to normalize the range of independent variables or features of
data.
# i.e.to standardize the data prior to performing classification
stand_scaler = StandardScaler()
x_train = stand_scaler.fit_transform(x_train)
x_test = stand_scaler.transform(x_test)
# Initialising KNN Classifier with value 3
classifier = KNeighborsClassifier(n_neighbors=3)
# Fitting the data to classifier
classifier.fit(x_train, y_train)
# Predict labels for x-test data
y_pred = classifier.predict(x_test)
print("\n Predicted Classs labels for testing data Y: ", y_pred)
# Confusion matrix with tested data
confusion_matrix_result = confusion_matrix(y_test, y_pred)
print("\n Confusion Matrix \n", confusion_matrix_result)
# total P+N can be calculated using sum(sum(conf_matrix))
total_value = sum(sum(confusion_matrix_result))
# Accuracy: TP+TN / P+N
```

```
Dataset:
[[ 1  0]
 [ 2  0]
 [ 3  0]
 [ 6  0]
 [ 6  0]
 [ 7  0]
[10  0]
[11  0]]
Size: 8

Class label for the datasets X Y: [0 0 1 1 1 0 0 0]

Training Data X:
[[ 1  0]
 [ 2  0]
 [ 3  0]
 [ 6  0]]

Training Data label Y: [0 0 1 1]

Predicted Classs labels for testing data Y: [1 1 1 1]

Confusion Matrix
[[0 3]
 [0 1]]

Accuracy : 0.25
Sensitivity : 1.0
Specificity : 0.0
```

Imported required libraries from library set. Declared X and y variables to take data in the arrays and then printing datasets. Then create labels for the datasets X and Y by taking test size and state values and then used scalar function to normalize the data and finally using KNN and taking n value as 3 calculated the accuracy, sensitivity and also the specifity values using the confusion matrix.

Source Code:

```
1. import numpy as nm
2. from sklearn.model_selection import train_test_split
3. from sklearn.preprocessing import StandardScaler
4. from sklearn.neighbors import KNeighborsClassifier
5. from sklearn.metrics import confusion_matrix
6. X = nm.array([[1, 0], [2, 0], [3, 0], [6, 0], [6, 0], [7, 0], [10, 0], [11, 0]])
7. print("Dataset: \n", X, "\n Size: ", len(X))
8. y = nm.array([0, 0, 1, 1, 1, 0, 0, 0])
```

```

9. print("\n Class label for the datasets X Y:", y)
10. x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=0, shuffle=False)
11. print("\nTraining Data X: \n", x_train)
12. print("\n Training Data label Y:", y_train) # feature Scaling-used to normalize the range of independent
    variables or features of data. to standardize the data prior to performing classification
13. stand_scalar = StandardScaler()
14. x_train = stand_scalar.fit_transform(x_train)
15. x_test = stand_scalar.transform(x_test) # Initialising KNN Classifier with value 3
16. classifier = KNeighborsClassifier(n_neighbors=3) # Fitting the data to classifier
17. classifier.fit(x_train, y_train) # Predict Labels for x-test data
18. y_pred = classifier.predict(x_test)
19. print("\nPredicted Classs labels for testing data Y:', y_pred) # Confusion matrix with tested data
20. confusion_matrix_result = confusion_matrix(y_test, y_pred)
21. print("\nConfusion Matrix\n", confusion_matrix_result) # total P+N can be calculated using
    sum(sum(conf_matrix))
22. total_value = sum(sum(confusion_matrix_result)) # Accuracy TN+TP / P+N
23. accuracy = (confusion_matrix_result[0, 0] + confusion_matrix_result[1, 1]) / total_value
24. print("\nAccuracy : ', accuracy) # Sensitivity TP/(TP+FN)
25. sensitivity = confusion_matrix_result[1, 1] / (confusion_matrix_result[1, 0] + confusion_matrix_result[1,
    1])
26. print('Sensitivity : ', sensitivity) # Specificity TN/(TN+FP)
27. specificity = confusion_matrix_result[0, 0] / (confusion_matrix_result[0, 0] + confusion_matrix_result[0,
    1])
28. print('Specificity : ', specificity)

```

Output:

Dataset:

```

[[ 1 0]
 [ 2 0]
 [ 3 0]
 [ 6 0]
 [ 6 0]
 [ 7 0]
 [10 0]
 [11 0]]
Size: 8

```

Class label for the datasets X Y: [0 0 1 1 1 0 0 0]

Training Data X:

```

[[1 0]
 [2 0]
 [3 0]
 [6 0]]

```

Training Data label Y: [0 0 1 1]

Predicted Classs labels for testing data Y: [1 1 1 1]

Confusion Matrix

```

[[0 3]
 [0 1]]

```

Accuracy : 0.25

Sensitivity : 1.0

Specificity : 0.0

Github: https://github.com/nagasurya99/ML_Assignment1

Video: <https://drive.google.com/file/d/17zmta6lN0hUcnEVLV46aeppu7LeNwN9Z/view?usp=sharing>