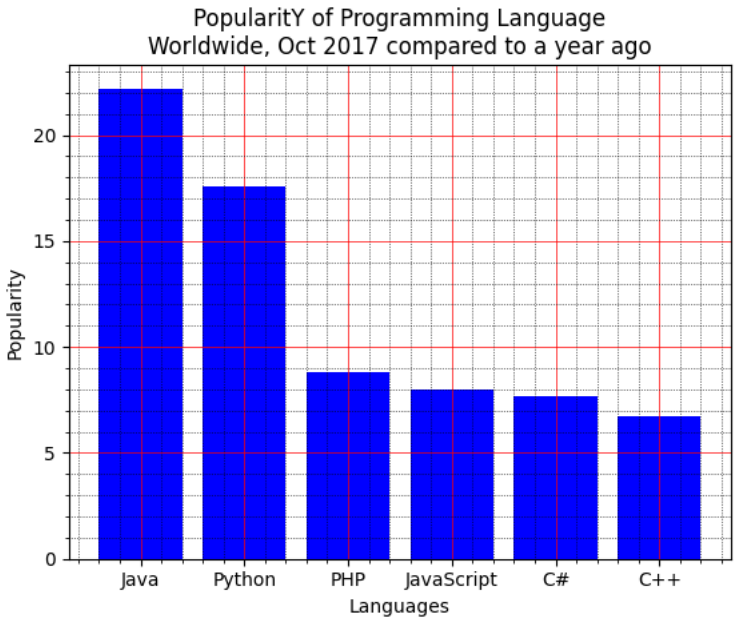
Write a Python programming to display a bar chart of the popularity of programming Languages.

Sample data:

Programming languages: Java, Python, PHP, JavaScript, C#, C++

Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.bar(x_pos, popularity, color='blue')
plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared to a year ago")
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

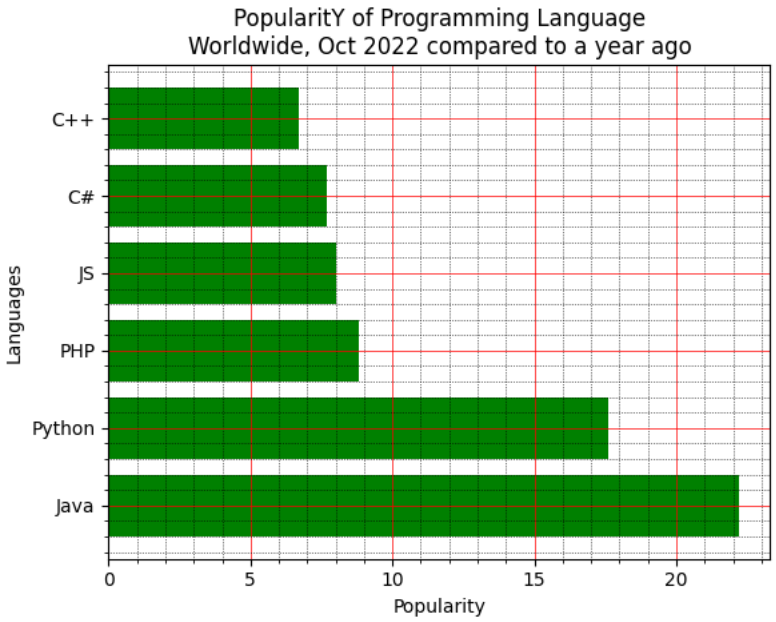


Write a Python programming to display a horizontal bar chart of the popularity of programming Languages.

Programming languages: Java, Python, PHP, JavaScript, C#, C++

Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JS', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.barh(x_pos, popularity, color='green')
plt.xlabel("Popularity")
plt.ylabel("Languages")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2022 compared to a year ago")
plt.yticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



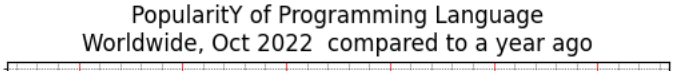
Write a Python programming to display a bar chart of the popularity of programming Languages.

Use different color for each bar.

Programming languages: Java, Python, PHP, JavaScript, C#, C++

Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, popularity, color=['red', 'black', 'green', 'blue', 'yellow', 'cyan'])

plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2022  compared to a year ago")
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

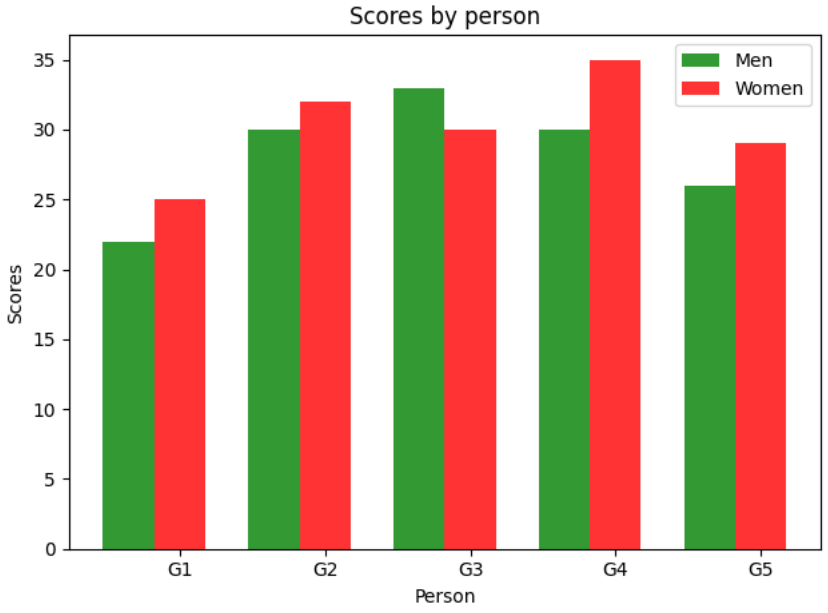


Write a Python program to create bar plot of scores by group and gender. Use multiple X values on the same chart for men and women.

Means (men) = (22, 30, 35, 35, 26)

Means (women) = (25, 32, 30, 35, 29)

```
import numpy as np
import matplotlib.pyplot as plt

# data to plot
n_groups = 5
men_means = (22, 30, 33, 30, 26)
women_means = (25, 32, 30, 35, 29)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, men_means, bar_width,
alpha=opacity,
color='g',
label='Men')

rects2 = plt.bar(index + bar_width, women_means, bar_width,
alpha=opacity,
color='r',
label='Women')

plt.xlabel('Person')
plt.ylabel('Scores')
plt.title('Scores by person')
plt.xticks(index + bar_width, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.legend()

plt.tight_layout()
plt.show()
```



Write a Python program to create bar plot from a DataFrame.

a b c d e

2 4,8,5,7,6

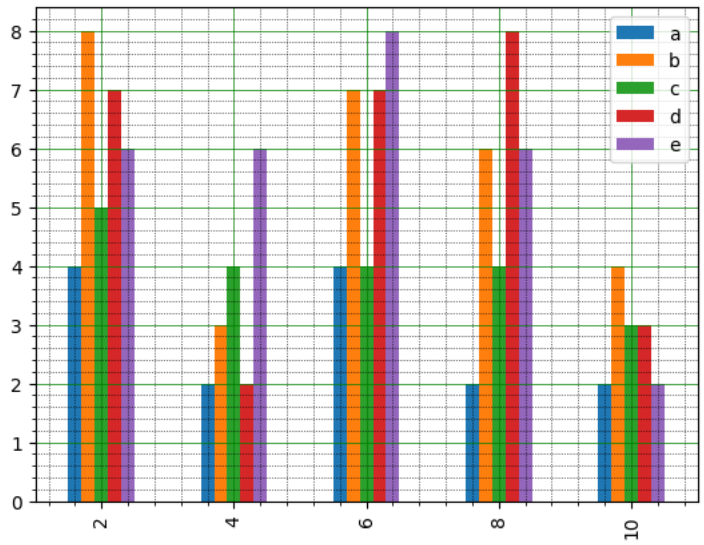
4 2,3,4,2,6

6 4,7,4,7,8

8 2,6,4,8,6

10 2,4,3,3,2

```
from pandas import DataFrame
import matplotlib.pyplot as plt
import numpy as np

a=np.array([[4,8,5,7,6],[2,3,4,2,6],[4,7,4,7,8],[2,6,4,8,6],[2,4,3,3,2]])
df=DataFrame(a, columns=['a','b','c','d','e'], index=[2,4,6,8,10])

df.plot(kind='bar')
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')

plt.show()
```



te a Python program to create a stacked bar plot with error bars.

Note: Use bottom to stack the women?s bars on top of the men?s bars.

Sample Data:

Means (men) = (22, 30, 35, 35, 26)

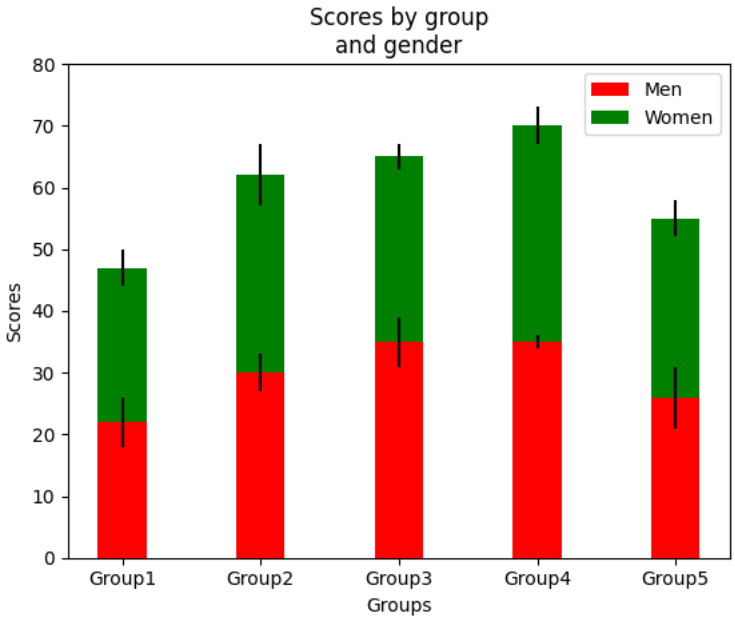
Means (women) = (25, 32, 30, 35, 29)

Men Standard deviation = (4, 3, 4, 1, 5)

Women Standard deviation = (3, 5, 2, 3, 3)

```
import numpy as np
import matplotlib.pyplot as plt

N = 5
menMeans = (22, 30, 35, 35, 26)
womenMeans = (25, 32, 30, 35, 29)
menStd = (4, 3, 4, 1, 5)
womenStd = (3, 5, 2, 3, 3)
# the x locations for the groups
ind = np.arange(N)
# the width of the bars
width = 0.35

p1 = plt.bar(ind, menMeans, width, yerr=menStd, color='red')
p2 = plt.bar(ind, womenMeans, width,
bottom=menMeans, yerr=womenStd, color='green')
```

```
plt.ylabel('Scores')
plt.xlabel('Groups')
plt.title('Scores by group\n' + 'and gender')
plt.xticks(ind, ('Group1', 'Group2', 'Group3', 'Group4', 'Group5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('Men', 'Women'))

plt.show()
```



Write a Python program to create stack bar plot and add label to each section.

people = ('G1','G2','G3','G4','G5','G6','G7','G8')

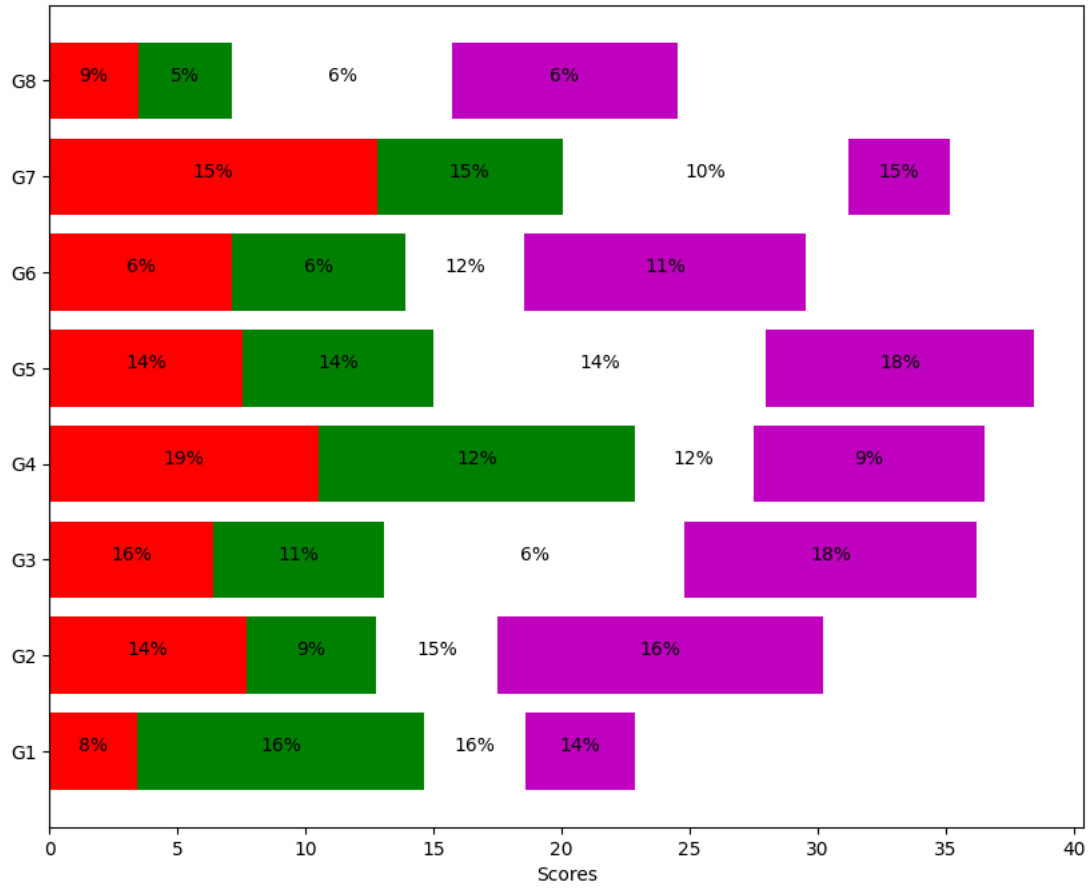
segments = 4

multi-dimensional data

data = [[3.40022085, 7.70632498, 6.4097905, 10.51648577, 7.5330039, 7.1123587, 12.77792868, 3.44773477], [11.24811149, 5.03778215, 6.65808464, 12.32220677, 7.45964195, 6.79685302, 7.24578743, 3.69371847], [3.94253354, 4.74763549, 11.73529246, 4.6465543, 12.9952182, 4.63832778, 11.16849999, 8.56883433], [4.24409799, 12.71746612, 11.3772169, 9.00514257, 10.47084185, 10.97567589, 3.98287652, 8.80552122]]

```
import numpy as np
import matplotlib.pyplot as plt

people = ('G1','G2','G3','G4','G5','G6','G7','G8')
segments = 4

# multi-dimensional data
data = [[  3.40022085,   7.70632498,   6.4097905,   10.51648577,   7.5330039,
    7.1123587,   12.77792868,   3.44773477],
 [ 11.24811149,   5.03778215,   6.65808464,  12.32220677,   7.45964195,
    6.79685302,   7.24578743,   3.69371847],
 [  3.94253354,   4.74763549,  11.73529246,   4.6465543,   12.9952182,
    4.63832778,  11.16849999,   8.56883433],
 [  4.24409799,  12.71746612,  11.3772169,    9.00514257,  10.47084185,
   10.97567589,   3.98287652,   8.80552122]]
percentages = (np.random.randint(5,20, (len(people), segments)))
y_pos = np.arange(len(people))

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111)

colors ='rgwm'
patch_handles = []
# left alignment of data starts at zero
```

```
left = np.zeros(len(people))
for i, d in enumerate(data):
    patch_handles.append(ax.barh(y_pos, d,
      color=colors[i%len(colors)], align='center',
      left=left))
    left += d

# search all of the bar segments and annotate
for j in range(len(patch_handles)):
    for i, patch in enumerate(patch_handles[j].get_children()):
        bl = patch.get_xy()
        x = 0.5*patch.get_width() + bl[0]
        y = 0.5*patch.get_height() + bl[1]
        ax.text(x,y, "%d%%" % (percentages[i,j]), ha='center')

ax.set_yticks(y_pos)
ax.set_yticklabels(people)
ax.set_xlabel('Scores')
plt.show()
```



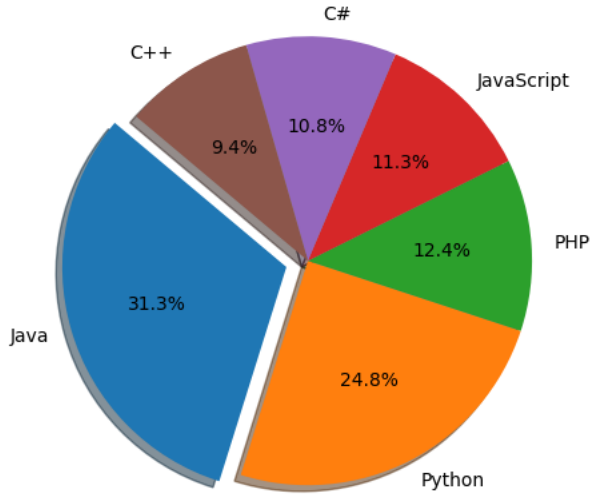
Write a Python programming to create a pie chart of the popularity of programming Languages.

Programming languages: Java, Python, PHP, JavaScript, C#, C++

Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

```
import matplotlib.pyplot as plt
# Data to plot
languages = 'Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++'
popurarity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b"]
# explode 1st slice
explode = (0.1, 0, 0, 0,0,0)
# Plot
plt.pie(popurarity, explode=explode, labels=languages, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
```

```
plt.show()
```

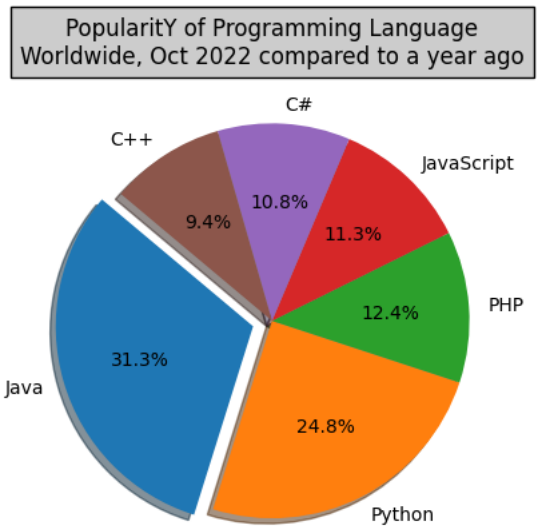


Write a Python programming to create a pie chart with a title of the popularity of programming Languages.

Programming languages: Java, Python, PHP, JavaScript, C#, C++

Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

```
import matplotlib.pyplot as plt
# Plot data
languages = 'Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++'
popuratity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
#colors = ['red', 'gold', 'yellowgreen', 'blue', 'lightcoral', 'lightskyblue']
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b"]
# explode 1st slice
explode = (0.1, 0, 0, 0, 0, 0)
# Plot
plt.pie(popuratity, explode=explode, labels=languages, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2022 compared to a year ago", bbox={'facecolor':'0.8', 'pad':5})
plt.show()
```



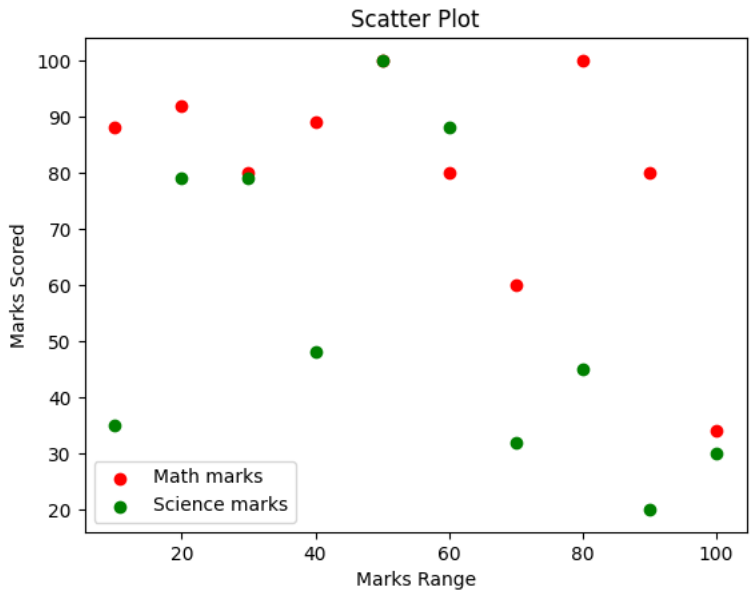
Write a Python program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10 students. Test Data:

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]

```
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()
```



Write a Python class to convert an integer to a Roman numeral.

```
class py_solution:
    def int_to_Roman(self, num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0
        while  num > 0:
            for _ in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
            i += 1
        return roman_num


print(py_solution().int_to_Roman(1))
print(py_solution().int_to_Roman(4000))
```

```
I
MMMM
```

Write a Python class to check the validity of a string of parentheses, '(', ')', '{', '}', '[' and ']. These brackets must be closed in the correct order, for example "()" and "()[]{}" are valid but "[)", "({[)]" and "{{{" are invalid.

```
class py_solution:
   def is_valid_parenthese(self, str1):
        stack, pchar = [], {"(": ")", "{": "}", "[": "]"}
        for parenthese in str1:
            if parenthese in pchar:
                stack.append(parenthese)
            elif len(stack) == 0 or pchar[stack.pop()] != parenthese:
                return False
        return len(stack) == 0

print(py_solution().is_valid_parenthese("(){}[]"))
print(py_solution().is_valid_parenthese("()[{)}"))
print(py_solution().is_valid_parenthese("()"))
```

```
True
False
True
```

Write a Python class Employee with attributes like emp_id, emp_name, emp_salary, and emp_department and methods like calculate_emp_salary, emp_assign_department, and print_employee_details.

Sample Employee Data:

"ADAMS", "E7876", 50000, "ACCOUNTING"

"JONES", "E7499", 45000, "RESEARCH"

"MARTIN", "E7900", 50000, "SALES"

"SMITH", "E7698", 55000, "OPERATIONS"

Use 'assign_department' method to change the department of an employee.

Use 'print_employee_details' method to print the details of an employee.

Use 'calculate_emp_salary' method takes two arguments: salary and hours_worked, which is the number of hours worked by the employee. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary.

Overtime is calculated as following formula:

overtime = hours_worked – 50

Overtime amount = (overtime * (salary / 50))

```
class Employee:
    def __init__(self, name, emp_id, salary, department):
        self.name = name
        self.id = emp_id
        self.salary = salary
        self.department = department

    def calculate_salary(self, salary, hours_worked):
        overtime = 0
        if hours_worked > 50:
            overtime = hours_worked - 50
        self.salary = self.salary + (overtime * (self.salary / 50))

    def assign_department(self, emp_department):
        self.department = emp_department

    def print_employee_details(self):
        print("\nName: ", self.name)
        print("ID: ", self.id)
        print("Salary: ", self.salary)
        print("Department: ", self.department)
        print("----------------------")


employee1 = Employee("ADAMS", "E7876", 50000, "ACCOUNTING")
employee2 = Employee("JONES", "E7499", 45000, "RESEARCH")
employee3 = Employee("MARTIN", "E7900", 50000, "SALES")
employee4 = Employee("SMITH", "E7698", 55000, "OPERATIONS")
```

```
print("Original Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
employee3.print_employee_details()
employee4.print_employee_details()

# Change the departments of employee1 and employee4
employee1.assign_department("OPERATIONS")
employee4.assign_department("SALES")

# Now calculate the overtime of the employees who are eligible:
employee2.calculate_salary(45000, 52)
employee4.calculate_salary(45000, 60)

print("Updated Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
employee3.print_employee_details()
employee4.print_employee_details()
```

```
Original Employee Details:

Name:  ADAMS
ID:  E7876
Salary:  50000
Department:  ACCOUNTING
----------------------

Name:  JONES
ID:  E7499
Salary:  45000
Department:  RESEARCH
----------------------

Name:  MARTIN
ID:  E7900
Salary:  50000
Department:  SALES
----------------------

Name:  SMITH
ID:  E7698
Salary:  55000
Department:  OPERATIONS
----------------------
Updated Employee Details:

Name:  ADAMS
ID:  E7876
Salary:  50000
Department:  OPERATIONS
----------------------

Name:  JONES
ID:  E7499
Salary:  46800.0
Department:  RESEARCH
----------------------

Name:  MARTIN
ID:  E7900
Salary:  50000
Department:  SALES
----------------------

Name:  SMITH
ID:  E7698
Salary:  66000.0
Department:  SALES
----------------------
```

Write a Python class Restaurant with attributes like menu_items, book_table, and customer_orders, and methods like add_item_to_menu, book_tables, and customer_order.

Perform the following tasks now:

Now add items to the menu.

Make table reservations.

Take customer orders.

Print the menu.

Print table reservations.

Print customer orders.

Note: Use dictionaries and lists to store the data.

```
class Restaurant:
    def __init__(self):
        self.menu_items = {}
        self.book_table = []
        self.customer_orders = []

    def add_item_to_menu(self, item, price):
        self.menu_items[item] = price

    def book_tables(self, table_number):
        self.book_table.append(table_number)

    def customer_order(self, table_number, order):
        order_details = {'table_number': table_number, 'order': order}
        self.customer_orders.append(order_details)

    def print_menu_items(self):
        for item, price in self.menu_items.items():
            print("{}: {}".format(item, price))

    def print_table_reservations(self):
        for table in self.book_table:
            print("Table {}".format(table))

    def print_customer_orders(self):
        for order in self.customer_orders:
            print("Table {}: {}".format(order['table_number'], order['order']))

restaurant = Restaurant()

# Add items
restaurant.add_item_to_menu("Cheeseburger", 9.99)
restaurant.add_item_to_menu("Caesar Salad", 8)
restaurant.add_item_to_menu("Grilled Salmon", 19.99)
restaurant.add_item_to_menu("French Fries", 3.99)
restaurant.add_item_to_menu("Fish & Chips:", 15)
# Book table
restaurant.book_tables(1)
restaurant.book_tables(2)
restaurant.book_tables(3)
# Order items
restaurant.customer_order(1, "Cheeseburger")
restaurant.customer_order(1, "Grilled Salmon")
restaurant.customer_order(2, "Fish & Chips")
restaurant.customer_order(2, "Grilled Salmon")

print("\nPopular dishes in the restaurant along with their prices:")
restaurant.print_menu_items()
print("\nTable reserved in the Restaurant:")
restaurant.print_table_reservations()
print("\nPrint customer orders:")
restaurant.print_customer_orders()
```

```
Popular dishes in the restaurant along with their prices:
Cheeseburger: 9.99
Caesar Salad: 8
Grilled Salmon: 19.99
French Fries: 3.99
Fish & Chips:: 15

Table reserved in the Restaurant:
Table 1
Table 2
Table 3

Print customer orders:
```

```
Table 1: Cheeseburger
Table 1: Grilled Salmon
Table 2: Fish & Chips
Table 2: Grilled Salmon
```

Write a Python class BankAccount with attributes like account_number, balance, date_of_opening and customer_name, and methods like deposit, withdraw, and check_balance.

```
class BankAccount:
    def __init__(self, account_number, date_of_opening, balance, customer_name):
        self.account_number = account_number
        self.date_of_opening  = date_of_opening
        self.balance = balance
        self.customer_name = customer_name

    def deposit(self, amount):
        self.balance += amount
        print(f"${amount} has been deposited in your account.")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance.")
        else:
            self.balance -= amount
            print(f"${amount} has been withdrawn from your account.")

    def check_balance(self):
        print(f"Current balance is ${self.balance}.")

    def print_customer_details(self):
        print("Name:", self.customer_name)
        print("Account Number:", self.account_number)
        print("Date of opening:", self.date_of_opening)
        print(f"Balance: ${self.balance}\n")

# Input customer details
ac_no_1 = BankAccount(2345, "01-01-2011", 1000, "Toninho Takeo")
ac_no_2 = BankAccount(1234, "11-03-2011", 2000, "Astrid Rugile")
ac_no_3 = BankAccount(2312, "12-01-2009", 3000, "Orli Kerenza")
ac_no_4 = BankAccount(1395, "01-01-2011", 3000, "Luciana Chika")
ac_no_5 = BankAccount(6345, "01-05-2011", 4000, "Toninho Takeo")

print("Customer Details:")
ac_no_1.print_customer_details()
ac_no_2.print_customer_details()
ac_no_3.print_customer_details()
ac_no_4.print_customer_details()
ac_no_5.print_customer_details()

print("============================")
ac_no_4.print_customer_details()
# Current balance is $3000.
# $1000 has been deposited in your account.
ac_no_4.deposit(1000)
ac_no_4.check_balance()
# Your current balance $4000.
# You want to withdraw $5000
ac_no_4.withdraw(5000)
# Output:
# Insufficient balance.
#The customer withdraw $3400.
ac_no_4.withdraw(3400)
ac_no_4.check_balance()
```

```
Customer Details:
Name: Toninho Takeo
Account Number: 2345
Date of opening: 01-01-2011
Balance: $1000

Name: Astrid Rugile
Account Number: 1234
Date of opening: 11-03-2011
Balance: $2000
```

```
Name: Orli Kerenza
Account Number: 2312
Date of opening: 12-01-2009
Balance: $3000

Name: Luciana Chika
Account Number: 1395
Date of opening: 01-01-2011
Balance: $3000

Name: Toninho Takeo
Account Number: 6345
Date of opening: 01-05-2011
Balance: $4000

============================
Name: Luciana Chika
Account Number: 1395
Date of opening: 01-01-2011
Balance: $3000

$1000 has been deposited in your account.
Current balance is $4000.
Insufficient balance.
$3400 has been withdrawn from your account.
Current balance is $600.
```

Write a Python class Inventory with attributes like item_id, item_name, stock_count, and price, and methods like add_item, update_item, and check_item_details.

Use a dictionary to store the item details, where the key is the item_id and the value is a dictionary containing the item_name, stock_count, and price

```
class Inventory:
    def __init__(self):
        self.inventory = {}
    def add_item(self, item_id, item_name, stock_count, price):
        self.inventory[item_id] = {"item_name": item_name, "stock_count": stock_count, "price": price}

    def update_item(self, item_id, stock_count, price):
        if item_id in self.inventory:
            self.inventory[item_id]["stock_count"] = stock_count
            self.inventory[item_id]["price"] = price
        else:
            print("Item not found in inventory.")

    def check_item_details(self, item_id):
        if item_id in self.inventory:
            item = self.inventory[item_id]
            return f"Product Name: {item['item_name']}, Stock Count: {item['stock_count']}, Price: {item['price']}"
        else:
            return "Item not found in inventory."

inventory = Inventory()

inventory.add_item("I001", "Laptop", 100, 500.00)
inventory.add_item("I002", "Mobile", 110, 450.00)
inventory.add_item("I003", "Desktop", 120, 500.00)
inventory.add_item("I004", "Tablet", 90, 550.00)
print("Item Details:")
print(inventory.check_item_details("I001"))
print(inventory.check_item_details("I002"))
print(inventory.check_item_details("I003"))
print(inventory.check_item_details("I004"))
print("\nUpdate the price of item code - 'I001':")
inventory.update_item("I001", 100, 505.00)
print(inventory.check_item_details("I001"))
print("\nUpdate the stock of item code - 'I003':")
inventory.update_item("I003", 115, 500.00)
print(inventory.check_item_details("I003"))
```

```
Item Details:
Product Name: Laptop, Stock Count: 100, Price: 500.0
Product Name: Mobile, Stock Count: 110, Price: 450.0
Product Name: Desktop, Stock Count: 120, Price: 500.0
Product Name: Tablet, Stock Count: 90, Price: 550.0
```

```
Update the price of item code - 'I001':
Product Name: Laptop, Stock Count: 100, Price: 505.0

Update the stock of item code - 'I003':
Product Name: Desktop, Stock Count: 115, Price: 500.0
```

Write a Python script to sort (ascending and descending) a dictionary by value

```
import operator
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('Original dictionary : ',d)
sorted_d = sorted(d.items(), key=operator.itemgetter(1))
print('Dictionary in ascending order by value : ',sorted_d)
sorted_d = dict( sorted(d.items(), key=operator.itemgetter(1),reverse=True))
print('Dictionary in descending order by value : ',sorted_d)
```

```
Original dictionary :  {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
Dictionary in ascending order by value :  [(0, 0), (2, 1), (1, 2), (4, 3), (3, 4)]
Dictionary in descending order by value :  {3: 4, 4: 3, 1: 2, 2: 1, 0: 0}
```

Write a Python program that calculates the area of a circle based on the radius entered by the user.

Write a Python program that accepts the user's first and last name and prints them in reverse order with a space between them.

```
fname = input("Input your First Name : ")
lname = input("Input your Last Name : ")
print ("Hello  " + lname + " " + fname)
```

Write a Python program that accepts a sequence of comma-separated numbers from the user and generates a list and a tuple of those numbers.

Sample data: 3, 5, 7, 23

```
values = input("Input some comma seprated numbers : ")
list = values.split(",")
tuple = tuple(list)
print('List : ',list)
print('Tuple : ',tuple)
```

```
Input some comma seprated numbers : 2 3 4 5 8 1 2 3 4
List :  ['2 3 4 5 8 1 2 3 4 ']
Tuple :  ('2 3 4 5 8 1 2 3 4 ',)
```

Write a Python program to display the first and last colors from the following list.

color_list = ["Red","Green","White" ,"Black"]

Double-click (or enter) to edit

```
color_list = ["Red","Green","White" ,"Black"]
print( "%s %s"%(color_list[0],color_list[-1]))
```

```
Red Black
```

Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn.

Write a Python program to create two empty classes, Student and Marks. Now create some instances and check whether they are instances of the said classes or not. Also, check whether the said classes are subclasses of the built-in object class or not.

```
class Student:
    pass
class Marks:
    pass
```

```
student1 = Student()
marks1 = Marks()
print(isinstance(student1, Student))
print(isinstance(marks1, Student))
print(isinstance(marks1, Marks))
print(isinstance(student1, Marks))
print("\nCheck whether the said classes are subclasses of the built-in object class or not.")
print(issubclass(Student, object))
print(issubclass(Marks, object))
```

Write a Python program to find the first appearance of the substrings 'not' and 'poor' in a given string. If 'not' follows 'poor', replace the whole 'not'...'poor' substring with 'good'. Return the resulting string.

```
def not_poor(str1):
  snot = str1.find('not')
  spoor = str1.find('poor')


  if spoor > snot and snot>0 and spoor>0:
    str1 = str1.replace(str1[snot:(spoor+4)], 'good')
    return str1
  else:
    return str1
print(not_poor('The lyrics is not that poor!'))
print(not_poor('The lyrics is poor!'))
```

```
 The lyrics is good!
 The lyrics is poor!
```

Write a Python program that accepts a comma-separated sequence of words as input and prints the distinct words in sorted form (alphanumerically).

```
items = input("Input comma separated sequence of words")
words = [word for word in items.split(",")]
print(",".join(sorted(list(set(words)))))
```

```
Input comma separated sequence of wordshello, i, we you
 i, we you,hello
```

Write a Python program to create a Caesar encryption.

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.

```
def caesar_encrypt(realText, step):
    outText = []
    cryptText = []

    uppercase = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
    lowercase = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',

    for eachLetter in realText:
        if eachLetter in uppercase:
            index = uppercase.index(eachLetter)
            crypting = (index + step) % 26
            cryptText.append(crypting)
            newLetter = uppercase[crypting]
            outText.append(newLetter)
        elif eachLetter in lowercase:
            index = lowercase.index(eachLetter)
            crypting = (index + step) % 26
            cryptText.append(crypting)
            newLetter = lowercase[crypting]
            outText.append(newLetter)
    return outText
```

```
code = caesar_encrypt('abc', 2)
print()
print(code)
print()
```

```
['c', 'd', 'e']
```

Write a Python program to count repeated characters in a string.

```
import collections
str1 = 'thequickbrownfoxjumpsoverthelazydog'
d = collections.defaultdict(int)
for c in str1:
    d[c] += 1

for c in sorted(d, key=d.get, reverse=True):
  if d[c] > 1:
      print('%s %d' % (c, d[c]))
```

```
o 4
e 3
t 2
h 2
u 2
r 2
```

Write a Python program to convert a given string into a list of words.

```
str1 = "The quick brown fox jumps over the lazy dog."
print(str1.split(' '))
str1 = "The-quick-brown-fox-jumps-over-the-lazy-dog."
print(str1.split('-'))
```

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog.']
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog.']
```

Write a Python program to find all the common characters in lexicographical order from two given lower case strings. If there are no similar letters print "No common characters".

```
from collections import Counter
def common_chars(str1,str2):
    d1 = Counter(str1)
    d2 = Counter(str2)
    common_dict = d1 & d2
    if len(common_dict) == 0:
        return "No common characters."

    # list of common elements
    common_chars = list(common_dict.elements())
    common_chars = sorted(common_chars)

    return ''.join(common_chars)

str1 = 'Python'
str2 = 'PHP'
print("Two strings: "+str1+' : '+str2)
print(common_chars(str1, str2))
str1 = 'Java'
str2 = 'PHP'
print("Two strings: "+str1+' : '+str2)
print(common_chars(str1, str2))
```

```
Two strings: Python : PHP
P
Two strings: Java : PHP
No common characters.
```

Write a Python program to check whether any word in a given string contains duplicate characters or not. Return True or False.

```
def duplicate_letters(text):
  word_list = text.split()
  for word in word_list:
    if len(word) > len(set(word)):
      return False
  return True
text = "Filter out the factorials of the said list."
print("Original text:")
print(text)
print("Check whether any word in the said sting contains duplicate characrters or not!")
print(duplicate_letters(text))
text = "Python Exercise."
print("\nOriginal text:")
print(text)
print("Check whether any word in the said sting contains duplicate characrters or not!")
print(duplicate_letters(text))
text = "The wait is over."
print("\nOriginal text:")
print(text)
print("Check whether any word in the said sting contains duplicate characrters or not!")
print(duplicate_letters(text))
```

```
Original text:
Filter out the factorials of the said list.
Check whether any word in the said sting contains duplicate characrters or not!
False

Original text:
Python Exercise.
Check whether any word in the said sting contains duplicate characrters or not!
False

Original text:
The wait is over.
Check whether any word in the said sting contains duplicate characrters or not!
True
```

Write a Python program to replace each character of a word of length five and more with a hash character (#).

Sample Output: Original string: Count the lowercase letters in the said list of words:

Replace words (length five or more) with hash characters in the said string:

the ######### ######## in the said list of

Original string: Python - Remove punctuations from a string:

Replace words (length five or more) with hash characters in the said string:

- ###### ############ from a

```
def test(text):
  for i in text.split():
    if len(i) >= 5:
      text = text.replace(i, "#" * len(i))
  return text

text ="Count the lowercase letters in the said list of words:"
print("Original string:", text)
print("Replace words (length five or more) with hash characters in the said string:")
print(test(text))
text = "Python - Remove punctuations from a string:"
print("\nOriginal string:", text)
print("Replace words (length five or more) with hash characters in the said string:")
print(test(text))
```

```
Original string: Count the lowercase letters in the said list of words:
Replace words (length five or more) with hash characters in the said string:
##### the ######### ######## in the said list of ######

Original string: Python - Remove punctuations from a string:
Replace words (length five or more) with hash characters in the said string:
###### - ###### ############ from a #######
```

Write a Python program to find out if a given array of integers contains any duplicate elements. Return true if any value appears at least twice in the array, and return false if every element is distinct

```
def test_duplicate(array_nums):
    nums_set = set(array_nums)
    return len(array_nums) != len(nums_set)
print(test_duplicate([1,2,3,4,5]))
print(test_duplicate([1,2,3,4, 4]))
print(test_duplicate([1,1,2,2,3,3,4,4,5]))
```

```
False
True
True
```

Write a Python program to check whether it follows the sequence given in the patterns array.

Pattern example: For color1 = ["red", "green", "green"] and patterns = ["a", "b", "b"]

the output should be samePatterns(color1, patterns) = true;

For color2 = ["red", "green", "greenn"] and patterns = ["a", "b", "b"]

the output should be samePatterns (strings, color2) = false.

```
def is_samePatterns(colors, patterns):
    if len(colors) != len(patterns):
        return False
    sdict = {}
    pset = set()
    sset = set()
    for i in range(len(patterns)):
        pset.add(patterns[i])
        sset.add(colors[i])
        if patterns[i] not in sdict.keys():
            sdict[patterns[i]] = []

        keys = sdict[patterns[i]]
        keys.append(colors[i])
        sdict[patterns[i]] = keys

    if len(pset) != len(sset):
        return False

    for values in sdict.values():

        for i in range(len(values) - 1):
            if values[i] != values[i+1]:
                return False

    return True

print(is_samePatterns(["red",
 "green",
 "green"], ["a",
 "b",
 "b"]))

print(is_samePatterns(["red",
 "green",
 "greenn"], ["a",
 "b",
 "b"]))
```

```
True
False
```

Write a Python program that removes all duplicate elements from an array and returns a new array.

```
import array as arr
def test(nums):
```

```
    return sorted(set(nums),key=nums.index)

array_num = arr.array('i', [1, 3, 5, 1, 3, 7, 9])
print("Original array:")
for i in range(len(array_num)):
    print(array_num[i], end=' ')
print("\nAfter removing duplicate elements from the said array:")
result = arr.array('i', test(array_num))
for i in range(len(result)):
    print(result[i], end=' ')
array_num = arr.array('i', [2, 4, 2, 6, 4, 8])
print("\nOriginal array:")
for i in range(len(array_num)):
    print(array_num[i], end=' ')
print("\nAfter removing duplicate elements from the said array:")
result = arr.array('i', test(array_num))
for i in range(len(result)):
    print(result[i], end=' ')
```

```
Original array:
1 3 5 1 3 7 9
After removing duplicate elements from the said array:
1 3 5 7 9
Original array:
2 4 2 6 4 8
After removing duplicate elements from the said array:
2 4 6 8
```

Write a Python program to find the missing number in a given array of numbers between 10 and 20.

```
import array as arr
def test(nums):
    return sum(range(10, 21)) - sum(list(nums))

array_num = arr.array('i', [10, 11, 12, 13, 14, 16, 17, 18, 19, 20])
print("Original array:")
for i in range(len(array_num)):
    print(array_num[i], end=' ')
print("\nMissing number in the said array (10-20): ",test(array_num))

array_num = arr.array('i', [10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
print("\nOriginal array:")
for i in range(len(array_num)):
    print(array_num[i], end=' ')
print("\nMissing number in the said array (10-20): ",test(array_num))
```

```
Original array:
10 11 12 13 14 16 17 18 19 20
Missing number in the said array (10-20):  15

Original array:
10 11 12 13 14 15 16 17 18 19
Missing number in the said array (10-20):  20
```

Write a Python function that accepts a string and counts the number of upper and lower case letters.

```
def string_test(s):
    d={"UPPER_CASE":0, "LOWER_CASE":0}
    for c in s:
        if c.isupper():
           d["UPPER_CASE"]+=1
        elif c.islower():
           d["LOWER_CASE"]+=1
        else:
           pass
    print ("Original String : ", s)
    print ("No. of Upper case characters : ", d["UPPER_CASE"])
    print ("No. of Lower case Characters : ", d["LOWER_CASE"])

string_test('The quick Brown Fox')
```

```
Original String :  The quick Brown Fox
No. of Upper case characters :  3
No. of Lower case Characters :  13
```

Write a Python function to check whether a number is "Perfect" or not

```
def perfect_number(n):
    sum = 0
    for x in range(1, n):
        if n % x == 0:
            sum += x
    return sum == n
print(perfect_number(6))
```

```
True
```

Write a Python function that prints out the first n rows of Pascal's triangle.

```
def pascal_triangle(n):
   trow = [1]
   y = [0]
   for x in range(max(n,0)):
      print(trow)
      trow=[l+r for l,r in zip(trow+y, y+trow)]
   return n>=1
pascal_triangle(6)
```

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
True
```

Write a Python function to create and print a list where the values are the squares of numbers between 1 and 30 (both included)

```
def printValues():
    l = list()
    for i in range(1,21):
        l.append(i**2)
    print(l)

printValues()
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]
```

Write a Python program to create a lambda function that adds 15 to a given number passed in as an argument, also create a lambda function that multiplies argument x with argument y and prints the result.

```
r = lambda a : a + 15
print(r(10))
r = lambda x, y : x * y
print(r(12, 4))
```

```
25
48
```

Write a Python program to sort a list of tuples using Lambda.

```
subject_marks = [('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]
print("Original list of tuples:")
print(subject_marks)
subject_marks.sort(key = lambda x: x[1])
print("\nSorting the List of Tuples:")
```

```
print(subject_marks)
```

```
Original list of tuples:
[('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]

Sorting the List of Tuples:
[('Social sciences', 82), ('English', 88), ('Science', 90), ('Maths', 97)]
```

Write a Python program to check whether a given string is a number or not using Lambda.

```
is_num = lambda q: q.replace('.','',1).isdigit()
print(is_num('26587'))
print(is_num('4.2365'))
print(is_num('-12547'))
print(is_num('00'))
print(is_num('A001'))
print(is_num('001'))
print("\nPrint checking numbers:")
is_num1 = lambda r: is_num(r[1:]) if r[0]=='-' else is_num(r)
print(is_num1('-16.4'))
print(is_num1('-24587.11'))
```

```
True
True
False
True
False
True

Print checking numbers:
True
True
```

Write a Python program to filter a given list to determine if the values in the list have a length of 6 using Lambda.

```
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
days = filter(lambda day: day if len(day)==6 else '', weekdays)
for d in days:
  print(d)
```

```
Monday
Friday
Sunday
```

Write a Python program to find the second lowest total marks of any student(s) from the given names and marks of each student using lists and lambda. Input the number of students, the names and grades of each student.

```
students = []
sec_name = []
second_low = 0
n = int(input("Input number of students: "))
for _ in range(n):
   s_name = input("Name: ")
   score = float(input("Grade: "))
   students.append([s_name,score])
print("\nNames and Grades of all students:")
print(students)
order =sorted(students, key = lambda x: int(x[1]))
for i in range(n):
   if order[i][1] != order[0][1]:
       second_low = order[i][1]
       break
print("\nSecond lowest grade: ",second_low)
sec_student_name = [x[0] for x in order if x[1] == second_low]
sec_student_name.sort()
print("\nNames:")
for s_name in sec_student_name:
   print(s_name)
```

```
Input number of students: 05
Name: nikhil
Grade: 50
Name: arun
Grade: 70
Name: bhushan
Grade: 78
Name: gopal
Grade: 67
Name: mukesh
Grade: 76

Names and Grades of all students:
[['nikhil', 50.0], ['arun', 70.0], ['bhushan', 78.0], ['gopal', 67.0], ['mukesh ', 76.0]]

Second lowest grade:  67.0

Names:
gopal
```

Write a Python program to extract the nth element from a given list of tuples using lambda.

```
def extract_nth_element(test_list, n):
    result = list(map (lambda x:(x[n]), test_list))
    return result
students = [('Greyson Fulton', 98, 99), ('Brady Kent', 97, 96), ('Wyatt Knott', 91, 94), ('Beau Turnbull', 94, 98)]
print ("Original list:")
print(students)
n = 0
print("\nExtract nth element ( n =",n,") from the said list of tuples:")
print(extract_nth_element(students, n))
n = 2
print("\nExtract nth element ( n =",n,") from the said list of tuples:")
print(extract_nth_element(students, n))
```

```
Original list:
[('Greyson Fulton', 98, 99), ('Brady Kent', 97, 96), ('Wyatt Knott', 91, 94), ('Beau Turnbull', 94, 98)]

Extract nth element ( n = 0 ) from the said list of tuples:
['Greyson Fulton', 'Brady Kent', 'Wyatt Knott', 'Beau Turnbull']

Extract nth element ( n = 2 ) from the said list of tuples:
[99, 96, 94, 98]
```

Double-click (or enter) to edit

✓ 0s completed at 3:02 AM

.1) Write a Pandas program to create a dataframe from a dictionary and display it

```
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]});
print(df)
```

```
    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

2) Write a Pandas program to create and display a DataFrame from a specified dictionary data which has the index labels.

```
import pandas as pd
import numpy as np

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
```

3)Write a Pandas program to get the first 3 rows of a given DataFrame

```
import pandas as pd
import numpy as np
```

```
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("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
```

4) Write a Pandas program to select the rows where the number of attempts in the examination is greater than 2.

```
import pandas as pd
import numpy as np

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("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
```

5) Write a Pandas program to select the rows the score is between 15 and 20 (inclusive)

```
import pandas as pd
import numpy as np
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("Rows where score between 15 and 20 (inclusive):")
print(df[df['score'].between(15, 20)])
```

```
Rows where score between 15 and 20 (inclusive):
        name  score  attempts qualify
c  Katherine   16.5         2     yes
f    Michael   20.0         3     yes
j      Jonas   19.0         1     yes
```

6) Write a Pandas program to calculate the mean of all students' scores. Data is stored in a dataframe.

```
import pandas as pd
import numpy as np
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("\nMean score for each different student in data frame:")
print(df['score'].mean())
```

```
Mean score for each different student in data frame:
13.5625
```

7) Write a Pandas program to append a new row 'k' to data frame with given values for each column. Now delete the new row and return the original DataFrame.

```
import pandas as pd
import numpy as np
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("Original rows:")
print(df)
print("\nAppend a new row:")
df.loc['k'] = [1, 'Suresh', 'yes', 15.5]
```

```
print("Print all records after insert a new record:")
print(df)
print("\nDelete the new row and display the original  rows:")
df = df.drop('k')
print(df)
```

```
Original rows:
        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

Append a new row:
Print all records after insert 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

Delete the new row and display the original  rows:
        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
```

8) Write a Pandas program to insert a new column in existing DataFrame.

```
import pandas as pd
import numpy as np
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("Original rows:")
print(df)
color = ['Red','Blue','Orange','Red','White','White','Blue','Green','Green','Red']
df['color'] = color
print("\nNew DataFrame after inserting the 'color' column")
print(df)
```

```
Original rows:
        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

New DataFrame after inserting the 'color' column
        name  score  attempts qualify   color
a  Anastasia   12.5         1     yes     Red
b       Dima    9.0         3      no    Blue
c  Katherine   16.5         2     yes  Orange
d      James    NaN         3      no     Red
e      Emily    9.0         2      no   White
f    Michael   20.0         3     yes   White
g    Matthew   14.5         1     yes    Blue
h      Laura    NaN         1      no   Green
i      Kevin    8.0         2      no   Green
j      Jonas   19.0         1     yes     Red
```

9) Write a Pandas program to select rows from a given DataFrame based on values in some columns

```
import pandas as pd
import numpy as np
d = {'col1': [1, 4, 3, 4, 5], 'col2': [4, 5, 6, 7, 8], 'col3': [7, 8, 9, 0, 1]}
df = pd.DataFrame(data=d)
print("Original DataFrame")
print(df)
print('Rows for colum1 value == 4')
print(df.loc[df['col1'] == 4])
```

```
Original DataFrame
   col1  col2  col3
0     1     4     7
1     4     5     8
2     3     6     9
3     4     7     0
4     5     8     1
Rows for colum1 value == 4
   col1  col2  col3
1     4     5     8
3     4     7     0
```

10) Write a Pandas program to set a given value for particular cell in DataFrame using index value

```
import pandas as pd
import numpy as np
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew
        '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']}
df = pd.DataFrame(exam_data)
print("Original DataFrame")
print(df)
print("\nSet a given value for particular cell in the DataFrame")
df.set_value(8, 'score', 10.2)
print(df)
```

11) Write a Pandas program to get the specified row value of a given DataFrame

```
import pandas as pd
d = {'col1': [1, 2, 3, 4, 7], 'col2': [4, 5, 6, 9, 5], 'col3': [7, 8, 12, 1, 11]}
df = pd.DataFrame(data=d)
print("Original DataFrame")
print(df)
print("Value of Row1")
print(df.iloc[0])
print("Value of Row4")
print(df.iloc[3])
```

```
Original DataFrame
   col1  col2  col3
0     1     4     7
1     2     5     8
2     3     6    12
3     4     9     1
4     7     5    11
Value of Row1
```

```
col1    1
col2    4
col3    7
Name: 0, dtype: int64
Value of Row4
col1    4
col2    9
col3    1
Name: 3, dtype: int64
```

12) Write a Pandas program to add a prefix or suffix to all columns of a given DataFrame

```
import pandas as pd
df = pd.DataFrame({'W':[68,75,86,80,66],'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97
print("Original DataFrame")
print(df)
print("\nAdd prefix:")
print(df.add_prefix("OO_"))
print("\nAdd suffix:")
print(df.add_suffix("_OYO"))
```

```
Original DataFrame
    W   X   Y   Z
0  68  78  84  86
1  75  85  94  97
2  86  96  89  96
3  80  80  83  72
4  66  86  86  83

Add prefix:
   OO_W  OO_X  OO_Y  OO_Z
0    68    78    84    86
1    75    85    94    97
2    86    96    89    96
3    80    80    83    72
4    66    86    86    83

Add suffix:
   W_OYO  X_OYO  Y_OYO  Z_OYO
0     68     78     84     86
1     75     85     94     97
2     86     96     89     96
3     80     80     83     72
4     66     86     86     83
```

13) Write a Pandas program to combine many given series to create a DataFrame.

```
import pandas as pd
sr1 = pd.Series(['php', 'python', 'java', 'c#', 'c++'])
sr2 = pd.Series([1, 2, 3, 4, 5])
```

```
print("Original Series:")
print(sr1)
print(sr2)
print("Combine above series to a dataframe:")
ser_df = pd.DataFrame(sr1, sr2).reset_index()
print(ser_df.head())
print("\nUsing pandas concat:")
ser_df = pd.concat([sr1, sr2], axis = 1)
print(ser_df.head())
print("\nUsing pandas DataFrame with a dictionary, gives a specific name to the columns:")
ser_df = pd.DataFrame({"col1":sr1, "col2":sr2})
print(ser_df.head(5))
```

```
Original Series:
0       php
1    python
2      java
3        c#
4       c++
dtype: object
0    1
1    2
2    3
3    4
4    5
dtype: int64
Combine above series to a dataframe:
   index       0
0      1  python
1      2    java
2      3      c#
3      4     c++
4      5     NaN

Using pandas concat:
        0  1
0     php  1
1  python  2
2    java  3
3      c#  4
4     c++  5

Using pandas DataFrame with a dictionary, gives a specific name to the columns:
     col1  col2
0     php     1
1  python     2
2    java     3
3      c#     4
4     c++     5
```

14) Write a Pandas program to fill missing values in time series data

```
import pandas as pd
import numpy as np
sdata = {"c1":[120, 130 ,140, 150, np.nan, 170], "c2":[7, np.nan, 10, np.nan, 5.5, 16.5]}
df = pd.DataFrame(sdata)
df.index = pd.util.testing.makeDateIndex()[0:6]
print("Original DataFrame:")
print(df)
print("\nDataFrame after interpolate:")
print(df.interpolate())
```

```
Original DataFrame:
               c1    c2
2000-01-03  120.0   7.0
2000-01-04  130.0   NaN
2000-01-05  140.0  10.0
2000-01-06  150.0   NaN
2000-01-07    NaN   5.5
2000-01-10  170.0  16.5

DataFrame after interpolate:
               c1     c2
2000-01-03  120.0   7.00
2000-01-04  130.0   8.50
2000-01-05  140.0  10.00
2000-01-06  150.0   7.75
2000-01-07  160.0   5.50
2000-01-10  170.0  16.50
/usr/local/lib/python3.9/dist-packages/pandas/util/__init__.py:16: FutureWarning: pandas
  import pandas.util.testing
```

15) Write a function division() that accepts two arguments. The function should be able to catch an exception such as ZeroDivisionError, ValueError, or any unknown error you might come across when you are doing a division operation.

```
def division(x,y):
    try:
        div = x/y
        print(f"{x}/{y} = {div}")
    except ZeroDivisionError as e:
        print("Exception occurred!:", e)
    except Exception as e:
        print("Exception occurred!:", e)
a = int(input("Enter number a: "))
b = int(input("Enter number b: "))
division(a,b)
```

```
Enter number a: 10
Enter number b: 0
Exception occurred!: division by zero
```

16) Write a function division() that accepts two arguments. The function should be able to catch an exception such as ZeroDivisionError, ValueError, or any unknown error you might come across when you are doing a division operation. Modify the earlier “perfect division function” task in such a way that the function division() has a clean-up action.

```
def division(x,y):
    try:
        div = x/y
        print(f"{x}/{y} = {div}")
    except ZeroDivisionError as e:
        print("Exception occurred!:", e)
    except Exception as e:
        print("Exception occurred!:", e)
    finally:
        print("Working on division function.")
a = int(input("Enter number a: "))
b = int(input("Enter number b: "))
division(a,b)
```

```
 Enter number a: 10
 Enter number b: 2
 10/2 = 5.0
 Working on division function.
```

17) Write a Python program to generate a random color hex, a random alphabetical string, random value between two integers (inclusive) and a random multiple of 7 between 0 and 70. Use random.randint()

```
import random
import string
print("Generate a random color hex:")
print("#{:06x}".format(random.randint(0, 0xFFFFFF)))
print("\nGenerate a random alphabetical string:")
max_length = 255
s = ""
for i in range(random.randint(1, max_length)):
    s += random.choice(string.ascii_letters)
print(s)
print("Generate a random value between two integers, inclusive:")
print(random.randint(0, 10))
print(random.randint(-7, 7))
print(random.randint(1, 1))
print("Generate a random multiple of 7 between 0 and 70:")
print(random.randint(0, 10) * 7)
```

```
Generate a random color hex:
#8476d8

Generate a random alphabetical string:
egecMJdxgWnTuFqiiqdHhlEqixIEwfcsNLOiwwctNMbvksPtSoDaolDJtmwsjKPxYjtJZufJyEsnrelMKIVcXyFr
Generate a random value between two integers, inclusive:
4
1
1
Generate a random multiple of 7 between 0 and 70:
49
```

18) Write a Python program that generates random alphabetical characters, alphabetical strings, and alphabetical strings of a fixed length. Use random.choice()

```
import random
import string
print("Generate a random alphabetical character:")
print(random.choice(string.ascii_letters))
print("\nGenerate a random alphabetical string:")
max_length = 255
str1 = ""
for i in range(random.randint(1, max_length)):
    str1 += random.choice(string.ascii_letters)
print(str1)
print("\nGenerate a random alphabetical string of a fixed length:")
str1 = ""
for i in range(10):
    str1 += random.choice(string.ascii_letters)
print(str1)
```

```
Generate a random alphabetical character:
m

Generate a random alphabetical string:
pgbFStqTZzEuxidLSUFvHKrUaZPRRGviEcHKWDsZmgAKqeRmOzGTRsHbBAOL

Generate a random alphabetical string of a fixed length:
owXGtCzxgn
```

19) Write a Python program to construct a seeded random number generator, also generate a float between 0 and 1, excluding 1. Use random.random()

```
import random
print("Construct a seeded random number generator:")
print(random.Random().random())
print(random.Random(0).random())
```

```
print("\nGenerate a float between 0 and 1, excluding 1:")
print(random.random())
```

```
Construct a seeded random number generator:
0.21531388930732276
0.8444218515250481

Generate a float between 0 and 1, excluding 1:
0.5916651103198761
```

20) Write a Python program to create a list of random integers and randomly select multiple items from the said list. Use random.sample()

```
import random
print("Create a list of random integers:")
population = range(0, 100)
nums_list = random.sample(population, 10)
print(nums_list)
no_elements = 4
print("\nRandomly select",no_elements,"multiple items from the said list:")
result_elements = random.sample(nums_list, no_elements)
print(result_elements)
no_elements = 8
print("\nRandomly select",no_elements,"multiple items from the said list:")
result_elements = random.sample(nums_list, no_elements)
print(result_elements)
```

```
Create a list of random integers:
[26, 34, 22, 87, 92, 94, 12, 75, 10, 82]

Randomly select 4 multiple items from the said list:
[10, 22, 12, 75]

Randomly select 8 multiple items from the said list:
[12, 22, 87, 26, 10, 75, 92, 94]
```

21) Write a Python program to check if a given value is a method of a user-defined class. Use types.MethodType()

```
import types
class C:
    def x():
        return 1
    def y():
        return 1
```

```
def b():
    return 2

print(isinstance(C().x, types.MethodType))
print(isinstance(C().y, types.MethodType))
print(isinstance(b, types.MethodType))
print(isinstance(max, types.MethodType))
print(isinstance(abs, types.MethodType))
```

```
True
True
False
False
False
```

22) Write a Python program to check if a given value is compiled code or not. Also check if a given value is a module or not. Use types.CodeType, types.ModuleType()

```
import types
print("Check if a given value is compiled code:")
code = compile("print('Hello')", "sample", "exec")
print(isinstance(code, types.CodeType))
print(isinstance("print(abs(-111))", types.CodeType))
print("\nCheck if a given value is a module:")
print(isinstance(types, types.ModuleType))
```

```
Check if a given value is compiled code:
True
False

Check if a given value is a module:
True
```

23) Write a Python program to construct a Decimal from a float and a Decimal from a string. Also represent the decimal value as a tuple. Use decimal.Decimal

```
import decimal
print("Construct a Decimal from a float:")
pi_val = decimal.Decimal(3.14159)
print(pi_val)
print(pi_val.as_tuple())
print("\nConstruct a Decimal from a string:")
num_str = decimal.Decimal("123.25")
print(num_str)
print(num_str.as_tuple())
```

```
Construct a Decimal from a float:
3.14158999999999988261834005243144929409027099609375
DecimalTuple(sign=0, digits=(3, 1, 4, 1, 5, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 8, 8, 2, 6,

Construct a Decimal from a string:
123.25
DecimalTuple(sign=0, digits=(1, 2, 3, 2, 5), exponent=-2)
```

24) Write a Python program to round a decimal value to the nearest multiple of 0.10, unless already an exact multiple of 0.05. Use decimal.Decimal

```
from decimal import Decimal


def round_to_10_cents(x):
    remainder = x.remainder_near(Decimal('0.10'))
    if abs(remainder) == Decimal('0.05'):
        return x
    else:
        return x - remainder

# Test code.
for x in range(80, 120):
    y = Decimal(x) / Decimal('1E2')
    print("{0} rounds to {1}".format(y, round_to_10_cents(y)))
```

```
0.80 rounds to 0.80
0.81 rounds to 0.80
0.82 rounds to 0.80
0.83 rounds to 0.80
0.84 rounds to 0.80
0.85 rounds to 0.85
0.86 rounds to 0.90
0.87 rounds to 0.90
0.88 rounds to 0.90
0.89 rounds to 0.90
0.90 rounds to 0.90
0.91 rounds to 0.90
0.92 rounds to 0.90
0.93 rounds to 0.90
0.94 rounds to 0.90
0.95 rounds to 0.95
0.96 rounds to 1.00
0.97 rounds to 1.00
0.98 rounds to 1.00
0.99 rounds to 1.00
1.00 rounds to 1.00
1.01 rounds to 1.00
1.02 rounds to 1.00
1.03 rounds to 1.00
```

```
1.04 rounds to 1.00
1.05 rounds to 1.05
1.06 rounds to 1.10
1.07 rounds to 1.10
1.08 rounds to 1.10
1.09 rounds to 1.10
1.10 rounds to 1.10
1.11 rounds to 1.10
1.12 rounds to 1.10
1.13 rounds to 1.10
1.14 rounds to 1.10
1.15 rounds to 1.15
1.16 rounds to 1.20
1.17 rounds to 1.20
1.18 rounds to 1.20
1.19 rounds to 1.20
```

25) Write a Python program to configure the rounding to round to the floor, ceiling. Use decimal.ROUND_FLOOR, decimal.ROUND_CEILING

```
import decimal
print("Configure the rounding to round to the floor:")
decimal.getcontext().prec = 4
decimal.getcontext().rounding = decimal.ROUND_FLOOR
print(decimal.Decimal(20) / decimal.Decimal(6))
print("\nConfigure the rounding to round to the ceiling:")
decimal.getcontext().prec = 4
decimal.getcontext().rounding = decimal.ROUND_CEILING
print(decimal.Decimal(20) / decimal.Decimal(6))
```

```
Configure the rounding to round to the floor:
3.333

Configure the rounding to round to the ceiling:
3.334
```

26) Write a Python program to configure rounding to round to the nearest integer, with ties going to the nearest even integer. Use decimal.ROUND_HALF_EVEN

```
import decimal
print("Configure the rounding to round to the nearest, with ties going to the nearest even in
decimal.getcontext().prec = 1
decimal.getcontext().rounding = decimal.ROUND_HALF_EVEN
print(decimal.Decimal(10) / decimal.Decimal(4))
```

```
Configure the rounding to round to the nearest, with ties going to the nearest even inte
2
```

27) Write a Python program to create a shallow copy of a given list. Use copy.copy

```
import copy
nums_x = [1, [2, 3, 4]]
print("Original list: ", nums_x)
nums_y = copy.copy(nums_x)
print("\nCopy of the said list:")
print(nums_y)
print("\nChange the value of an element of the original list:")
nums_x[1][1] = 10
print(nums_x)
print("\nSecond list:")
print(nums_y)
nums =  [[1], [2]]
nums_copy = copy.copy(nums)
print("\nOriginal list:")
print(nums)
print("\nCopy of the said list:")
print(nums_copy)
print("\nChange the value of an element of the original list:")
nums[0][0] = 0
print("\nFirst list:")
print(nums)
print("\nSecond list:")
print(nums_copy)
```

```
Original list:  [1, [2, 3, 4]]

Copy of the said list:
[1, [2, 3, 4]]

Change the value of an element of the original list:
[1, [2, 10, 4]]

Second list:
[1, [2, 10, 4]]

Original list:
[[1], [2]]

Copy of the said list:
[[1], [2]]

Change the value of an element of the original list:

First list:
[[0], [2]]
```

```
Second list:
[[0], [2]]
```

27) Write a Python program to create a deep copy of a given list. Use copy.copy

```
import copy
nums_x = [1, [2, 3, 4]]
print("Original list: ", nums_x)
nums_y = copy.deepcopy(nums_x)
print("\nDeep copy of the said list:")
print(nums_y)
print("\nChange the value of an element of the original list:")
nums_x[1][1] = 10
print(nums_x)
print("\nCopy of the second list (Deep copy):")
print(nums_y)
nums = [[1, 2, 3], [4, 5, 6]]
deep_copy = copy.deepcopy(nums)
print("\nOriginal list:")
print(nums)
print("\nDeep copy of the said list:")
print(deep_copy)
print("\nChange the value of some elements of the original list:")
nums[0][2] = 55
nums[1][1] = 77
print("\nOriginal list:")
print(nums)
print("\nSecond list (Deep copy):")
print(deep_copy)
```

```
Original list:  [1, [2, 3, 4]]

Deep copy of the said list:
[1, [2, 3, 4]]

Change the value of an element of the original list:
[1, [2, 10, 4]]

Copy of the second list (Deep copy):
[1, [2, 3, 4]]

Original list:
[[1, 2, 3], [4, 5, 6]]

Deep copy of the said list:
[[1, 2, 3], [4, 5, 6]]

Change the value of some elements of the original list:

Original list:
[[1, 2, 55], [4, 77, 6]]
```

```
Second list (Deep copy):
[[1, 2, 3], [4, 5, 6]]
```

28) Write a Python program to create a shallow copy of a given dictionary. Use copy.copy

```
import copy
nums_x = {"a":1, "b":2, 'cc':{"c":3}}
print("Original dictionary: ", nums_x)
nums_y = copy.copy(nums_x)
print("\nCopy of the said list:")
print(nums_y)
print("\nChange the value of an element of the original dictionary:")
nums_x["cc"]["c"] = 10
print(nums_x)
print("\nSecond dictionary:")
print(nums_y)

nums = {"x":1, "y":2, 'zz':{"z":3}}
nums_copy = copy.copy(nums)
print("\nOriginal dictionary :")
print(nums)
print("\nCopy of the said list:")
print(nums_copy)
print("\nChange the value of an element of the original dictionary:")
nums["zz"]["z"] = 10
print("\nFirst dictionary:")
print(nums)
print("\nSecond dictionary (copy):")
print(nums_copy)
```

```
Original dictionary:  {'a': 1, 'b': 2, 'cc': {'c': 3}}

Copy of the said list:
{'a': 1, 'b': 2, 'cc': {'c': 3}}

Change the value of an element of the original dictionary:
{'a': 1, 'b': 2, 'cc': {'c': 10}}

Second dictionary:
{'a': 1, 'b': 2, 'cc': {'c': 10}}

Original dictionary :
{'x': 1, 'y': 2, 'zz': {'z': 3}}

Copy of the said list:
{'x': 1, 'y': 2, 'zz': {'z': 3}}

Change the value of an element of the original dictionary:

First dictionary:
```

```
{'x': 1, 'y': 2, 'zz': {'z': 10}}

Second dictionary (copy):
{'x': 1, 'y': 2, 'zz': {'z': 10}}
```

29) Write a Python program to create a deep copy of a given dictionary. Use copy.copy

```
import copy
nums_x = {"a":1, "b":2, 'cc':{"c":3}}
print("Original dictionary: ", nums_x)
nums_y = copy.deepcopy(nums_x)
print("\nDeep copy of the said list:")
print(nums_y)
print("\nChange the value of an element of the original dictionary:")
nums_x["cc"]["c"] = 10
print(nums_x)
print("\nSecond dictionary (Deep copy):")
print(nums_y)

nums = {"x":1, "y":2, 'zz':{"z":3}}
nums_copy = copy.deepcopy(nums)
print("\nOriginal dictionary :")
print(nums)
print("\nDeep copy of the said list:")
print(nums_copy)
print("\nChange the value of an element of the original dictionary:")
nums["zz"]["z"] = 10
print("\nFirst dictionary:")
print(nums)
print("\nSecond dictionary (Deep copy):")
print(nums_copy)
```

```
Original dictionary:  {'a': 1, 'b': 2, 'cc': {'c': 3}}

Deep copy of the said list:
{'a': 1, 'b': 2, 'cc': {'c': 3}}

Change the value of an element of the original dictionary:
{'a': 1, 'b': 2, 'cc': {'c': 10}}

Second dictionary (Deep copy):
{'a': 1, 'b': 2, 'cc': {'c': 3}}

Original dictionary :
{'x': 1, 'y': 2, 'zz': {'z': 3}}

Deep copy of the said list:
{'x': 1, 'y': 2, 'zz': {'z': 3}}

Change the value of an element of the original dictionary:
```

```
First dictionary:
{'x': 1, 'y': 2, 'zz': {'z': 10}}

Second dictionary (Deep copy):
{'x': 1, 'y': 2, 'zz': {'z': 3}}
```

30) Write a Python GUI program to create a label and change the label font style (font name, bold, size) using tkinter module

```
import tkinter as tk
parent = tk.Tk()
parent.title("-Welcome to Python tkinter Basic exercises-")
my_label = tk.Label(parent, text="Hello", font=("Arial Bold", 70))
my_label.grid(column=0, row=0)
parent.mainloop()
```

31) Write a Python GUI program to create a window and disable to resize the window using tkinter module.

```
import tkinter as tk
parent = tk.Tk()
parent.title("-Welcome to Python tkinter Basic exercises-")
# Disable resizing the GUI
parent.resizable(0,0)
parent.mainloop()
```

32) Write a Python GUI program to create two buttons exit and hello using tkinter module.

```
import tkinter as tk

def write_text():
    print("Tkinter is easy to create GUI!")

parent = tk.Tk()
frame = tk.Frame(parent)
frame.pack()

text_disp= tk.Button(frame,
                   text="Hello",
                   command=write_text
                   )

text_disp.pack(side=tk.LEFT)
```

```
exit_button = tk.Button(frame,
                   text="Exit",
                   fg="green",
                   command=quit)
exit_button.pack(side=tk.RIGHT)

parent.mainloop()
```

33) Write a Python GUI program to create a Text widget using tkinter module. Insert a string at the beginning then insert a string into the current text. Delete the first and last character of the text.

```
import tkinter as tk

parent = tk.Tk()
# create the widget.
mytext = tk.Text(parent)

# insert a string at the beginning
mytext.insert('1.0', "- Python exercises, solution -")

# insert a string into the current text
mytext.insert('1.19', ' Practice,')

# delete the first and last character (including a newline character)
mytext.delete('1.0')
mytext.delete('end - 2 chars')
mytext.pack()
parent.mainloop()
```

34) Write a Python GUI program to create a Progress bar widgets using tkinter module.

```
import tkinter as tk
from tkinter.ttk import Progressbar
from tkinter import ttk
parent = tk.Tk()
parent.title("Progressbar")
parent.geometry('350x200')
style = ttk.Style()
style.theme_use('default')
style.configure("black.Horizontal.TProgressbar", background='green')
bar = Progressbar(parent, length=220, style='black.Horizontal.TProgressbar')
bar['value'] = 80
bar.grid(column=0, row=0)
parent.mainloop()
```

35)Write a Python GUI program to create a Progress bar widgets using tkinter module.

```
import tkinter as tk
from tkinter.ttk import Progressbar
from tkinter import ttk
parent = tk.Tk()
parent.title("Progressbar")
parent.geometry('350x200')
style = ttk.Style()
style.theme_use('default')
style.configure("black.Horizontal.TProgressbar", background='green')
bar = Progressbar(parent, length=220, style='black.Horizontal.TProgressbar')
bar['value'] = 80
bar.grid(column=0, row=0)
parent.mainloop()
```

36) Write a Python GUI program to create three radio buttons widgets using tkinter module

```
import tkinter as tk
parent = tk.Tk()
parent.title("Radiobutton ")
parent.geometry('350x200')
radio1 = tk.Radiobutton(parent, text='First', value=1)
radio2 = tk.Radiobutton(parent, text='Second', value=2)
radio3 = tk.Radiobutton(parent, text='Third', value=3)
radio1.grid(column=0, row=0)
radio2.grid(column=1, row=0)
```

36) Write a Python GUI program to create a Spinbox widget using tkinter module

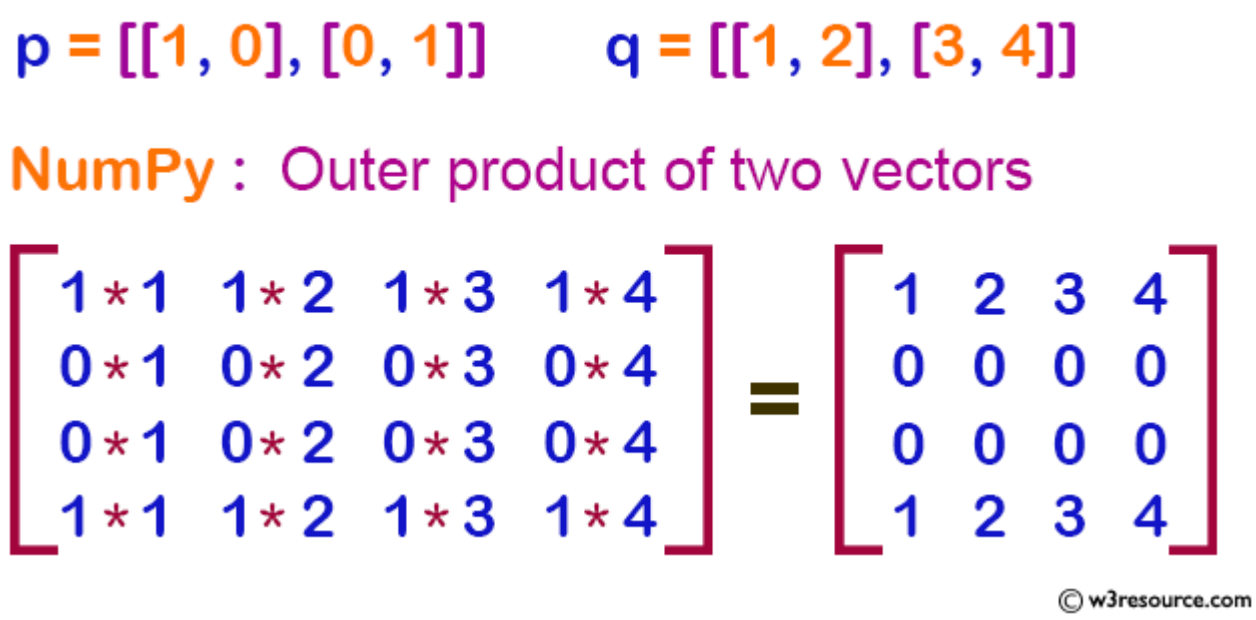
```
import tkinter as tk
root = tk.Tk()
text_var = tk.DoubleVar()

spin_box = tk.Spinbox(
    root,
    from_=0.6,
    to=50.0,
    increment=.01,
    textvariable=text_var
)
spin_box.pack()
root.mainloop()
```

37) Write a NumPy program to compute the multiplication of two given matrixes

```
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("original matrix:")
print(p)
print(q)
result1 = np.dot(p, q)
print("Result of the said matrix multiplication:")
print(result1)
```

```
original matrix:
[[1, 0], [0, 1]]
[[1, 2], [3, 4]]
Result of the said matrix multiplication:
[[1 2]
 [3 4]]
```

38) Write a NumPy program to compute the outer product of two given vectors.




```
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("original matrix:")
print(p)
print(q)
result = np.outer(p, q)
print("Outer product of the said two vectors:")
```

```
print(result)
```

```
original matrix:
[[1, 0], [0, 1]]
[[1, 2], [3, 4]]
Outer product of the said two vectors:
[[1 2 3 4]
 [0 0 0 0]
 [0 0 0 0]
 [1 2 3 4]]
```

39) Write a NumPy program to evaluate Einstein's summation convention of two given multidimensional arrays.

Note: In mathematics, especially in applications of linear algebra to physics, the Einstein notation or Einstein summation convention is a notational convention that implies summation over a set of indexed terms in a formula, thus achieving notational brevity.

```
import numpy as np
a = np.array([1,2,3])
b = np.array([0,1,0])
print("Original 1-d arrays:")
print(a)
print(b)
result =  np.einsum("n,n", a, b)
print("Einstein's summation convention of the said arrays:")
print(result)
x = np.arange(9).reshape(3, 3)
y = np.arange(3, 12).reshape(3, 3)
print("Original Higher dimension:")
print(x)
print(y)
result = np.einsum("mk,kn", x, y)
print("Einstein's summation convention of the said arrays:")
print(result)
```

```
Original 1-d arrays:
[1 2 3]
[0 1 0]
Einstein's summation convention of the said arrays:
2
Original Higher dimension:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Einstein's summation convention of the said arrays:
```

```
[[ 24  27  30]
 [ 78  90 102]
 [132 153 174]]
```

40) Write a NumPy program to compute the determinant of an array.

```
import numpy as np
a = np.array([[1,2],[3,4]])
print("Original array:")
print(a)
result =  np.linalg.det(a)
print("Determinant of the said array:")
print(result)
```

```
Original array:
[[1 2]
 [3 4]]
Determinant of the said array:
-2.0000000000000004
```

41) Write a NumPy program to compute the sum of the diagonal element of a given array

```
import numpy as np
m = np.arange(6).reshape(2,3)
print("Original matrix:")
print(m)
result =  np.trace(m)
print("Condition number of the said matrix:")
print(result)
```

```
Original matrix:
[[0 1 2]
 [3 4 5]]
Condition number of the said matrix:
4
```

42) Write a NumPy program to compute the factor of a given array by Singular Value Decomposition.

```
import numpy as np
a = np.array([[1, 0, 0, 0, 2], [0, 0, 3, 0, 0], [0, 0, 0, 0, 0], [0, 2, 0, 0, 0]], dtype=np.f
print("Original array:")
print(a)
U, s, V = np.linalg.svd(a, full_matrices=False)
q, r = np.linalg.qr(a)
```

```
print("Factor of a given array  by Singular Value Decomposition:")
print("U=\n", U, "\ns=\n", s, "\nV=\n", V)
```

```
Original array:
[[1. 0. 0. 0. 2.]
 [0. 0. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0.]]
Factor of a given array  by Singular Value Decomposition:
U=
 [[ 0.  1.  0.  0.]
 [ 1.  0.  0.  0.]
 [ 0.  0.  0. -1.]
 [ 0.  0.  1.  0.]]
s=
 [3.       2.236068 2.       0.      ]
V=
 [[-0.         0.         1.        -0.         0.       ]
 [ 0.4472136  0.         0.         0.         0.8944272]
 [-0.         1.         0.        -0.         0.       ]
 [ 0.         0.         0.         1.         0.       ]]
```

43) Write a NumPy program to calculate the Frobenius norm and the condition number of a given array.

```
import numpy as np
a = np.arange(1, 10).reshape((3, 3))
print("Original array:")
print(a)
print("Frobenius norm and the condition number:")
print(np.linalg.norm(a, 'fro'))
print(np.linalg.cond(a, 'fro'))
```

```
Original array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Frobenius norm and the condition number:
16.881943016134134
inf
```

44) Write a NumPy program to reverse an array (the first element becomes the last)

```
import numpy as np
import numpy as np
x = np.arange(12, 38)
print("Original array:")
print(x)
```

```
print("Reverse array:")
x = x[::-1]
print(x)
```

```
Original array:
[12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
 36 37]
Reverse array:
[37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14
 13 12]
```

45) Write a NumPy program to create an 8x8 matrix and fill it with a checkerboard pattern

```
import numpy as np
x = np.ones((3,3))
print("Checkerboard pattern:")
x = np.zeros((8,8),dtype=int)
x[1::2,::2] = 1
x[::2,1::2] = 1
print(x)
```

```
Checkerboard pattern:
[[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

46) Write a NumPy program to convert a list and tuple into arrays.

```
import numpy as np
my_list = [1, 2, 3, 4, 5, 6, 7, 8]
print("List to array: ")
print(np.asarray(my_list))
my_tuple = ([8, 4, 6], [1, 2, 3])
print("Tuple to array: ")
print(np.asarray(my_tuple))
```

```
List to array:
[1 2 3 4 5 6 7 8]
Tuple to array:
[[8 4 6]
 [1 2 3]]
```

47) Write a NumPy program to convert Centigrade degrees into Fahrenheit degrees. Centigrade values are stored in a NumPy array

```
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.  ]
```

48) Write a NumPy program to find the set difference between two arrays. The set difference will return sorted, distinct values in array1 that are not in array2.

```
import numpy as np
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:")
print(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]
```

49) Write a NumPy program to create a 5x5 matrix with row values ranging from 0 to 4

```
import numpy as np
x = np.zeros((5,5))
print("Original array:")
print(x)
print("Row values ranging from 0 to 4.")
x += np.arange(5)
print(x)
```

```
Original array:
[[0. 0. 0. 0. 0.]
```

```
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
Row values ranging from 0 to 4.
[[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
```

50) Write a Python program to read an entire text file.

```
def file_read(fname):
        txt = open(fname)
        print(txt.read())

file_read('test.txt')
```

51) Write a Python program to read a file line by line and store it into a list

```
def file_read(fname):
        with open(fname) as f:
                #Content_list is the list that contains the read lines.
                content_list = f.readlines()
                print(content_list)

file_read(\'test.txt\')
```

52) Write a Python program to count the number of lines in a text file

```
def file_lengthy(fname):
        with open(fname) as f:
                for i, l in enumerate(f):
                        pass
        return i + 1
print("Number of lines in the file: ",file_lengthy("test.txt"))
```

53) Write a Python program to copy the contents of a file to another file

```
from shutil import copyfile
copyfile('test.py', 'abc.py')
```

54) Write a Python program that takes a text file as input and returns the number of words of a given text file.

```
def count_words(filepath):
   with open(filepath) as f:
       data = f.read()
       data.replace(",", " ")
       return len(data.split(" "))
print(count_words("words.txt"))
```

55) Write a Python program to create a file where all letters of English alphabet are listed by specified number of letters on each line.

```
import string
def letters_file_line(n):
   with open("words1.txt", "w") as f:
       alphabet = string.ascii_uppercase
       letters = [alphabet[i:i + n] + "\n" for i in range(0, len(alphabet), n)]
       f.writelines(letters)
letters_file_line(3)
```

56) Write a Pandas program to merge two given datasets using multiple join keys

```
import pandas as pd
data1 = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                      'key2': ['K0', 'K1', 'K0', 'K1'],
                      'P': ['P0', 'P1', 'P2', 'P3'],
                      'Q': ['Q0', 'Q1', 'Q2', 'Q3']})
data2 = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                       'key2': ['K0', 'K0', 'K0', 'K0'],
                       'R': ['R0', 'R1', 'R2', 'R3'],
                       'S': ['S0', 'S1', 'S2', 'S3']})
print("Original DataFrames:")
print(data1)
print("--------------------")
print(data2)
print("\nMerged Data:")
merged_data = pd.merge(data1, data2, on=['key1', 'key2'])
print(merged_data)
```

```
Original DataFrames:
  key1 key2   P   Q
0   K0   K0  P0  Q0
1   K0   K1  P1  Q1
2   K1   K0  P2  Q2
3   K2   K1  P3  Q3
--------------------
  key1 key2   R   S
0   K0   K0  R0  S0
1   K1   K0  R1  S1
2   K1   K0  R2  S2
3   K2   K0  R3  S3

Merged Data:
  key1 key2   P   Q   R   S
0   K0   K0  P0  Q0  R0  S0
1   K1   K0  P2  Q2  R1  S1
2   K1   K0  P2  Q2  R2  S2
```

57) Write a Pandas program to Combine two DataFrame objects by filling null values in one DataFrame with non-null values from other DataFrame.

```
import pandas as pd
df1 = pd.DataFrame({'A': [None, 0, None], 'B': [3, 4, 5]})
df2 = pd.DataFrame({'A': [1, 1, 3], 'B': [3, None, 3]})
df1.combine_first(df2)
print("Original DataFrames:")
print(df1)
print("--------------------")
print(df2)
print("\nMerge two dataframes with different columns:")
result = df1.combine_first(df2)
print(result)
```

```
Original DataFrames:
     A  B
0  NaN  3
1  0.0  4
2  NaN  5
--------------------
   A    B
0  1  3.0
1  1  NaN
2  3  3.0

Merge two dataframes with different columns:
     A    B
0  1.0  3.0
1  0.0  4.0
2  3.0  5.0
```

58) Write a Pandas program to join the two dataframes using the common column of both dataframes.

```
import pandas as pd
student_data1 = pd.DataFrame({
        'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'],
         'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Mor
        'marks': [200, 210, 190, 222, 199]})

student_data2 = pd.DataFrame({
        'student_id': ['S4', 'S5', 'S6', 'S7', 'S8'],
        'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Ma
        'marks': [201, 200, 198, 219, 201]})

print("Original DataFrames:")
print(student_data1)
print(student_data2)
merged_data = pd.merge(student_data1, student_data2, on='student_id', how='inner')
print("Merged data (inner join):")
print(merged_data)
```

```
Original DataFrames:
  student_id              name  marks
0         S1  Danniella Fenton    200
1         S2      Ryder Storey    210
2         S3      Bryce Jensen    190
3         S4         Ed Bernal    222
4         S5       Kwame Morin    199
  student_id              name  marks
0         S4  Scarlette Fisher    201
1         S5  Carla Williamson    200
2         S6       Dante Morse    198
```

```
3         S7    Kaiser William    219
4         S8   Madeeha Preston    201
Merged data (inner join):
  student_id       name_x  marks_x            name_y  marks_y
0         S4    Ed Bernal      222  Scarlette Fisher      201
1         S5  Kwame Morin      199  Carla Williamson      200
```

59) Write a Pandas program to join (left join) the two dataframes using keys from left dataframe only.

```
import pandas as pd
data1 = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                      'key2': ['K0', 'K1', 'K0', 'K1'],
                      'P': ['P0', 'P1', 'P2', 'P3'],
                      'Q': ['Q0', 'Q1', 'Q2', 'Q3']})
data2 = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                      'key2': ['K0', 'K0', 'K0', 'K0'],
                      'R': ['R0', 'R1', 'R2', 'R3'],
                      'S': ['S0', 'S1', 'S2', 'S3']})
print("Original DataFrames:")
print(data1)
print("--------------------")
print(data2)
print("\nMerged Data (keys from data1):")
merged_data = pd.merge(data1, data2, how='left', on=['key1', 'key2'])
print(merged_data)
print("\nMerged Data (keys from data2):")
merged_data = pd.merge(data2, data1, how='left', on=['key1', 'key2'])
print(merged_data)
```

```
Original DataFrames:
  key1 key2   P   Q
0   K0   K0  P0  Q0
1   K0   K1  P1  Q1
2   K1   K0  P2  Q2
3   K2   K1  P3  Q3
--------------------
  key1 key2   R   S
0   K0   K0  R0  S0
1   K1   K0  R1  S1
2   K1   K0  R2  S2
3   K2   K0  R3  S3

Merged Data (keys from data1):
  key1 key2   P   Q    R    S
0   K0   K0  P0  Q0   R0   S0
1   K0   K1  P1  Q1  NaN  NaN
2   K1   K0  P2  Q2   R1   S1
3   K1   K0  P2  Q2   R2   S2
4   K2   K1  P3  Q3  NaN  NaN
```

✓ 0s completed at 3:36 AM